ISO TC184/SC4/ WG3 N 701 (P ___)

Date: November 21, 1997 Supersedes SC4/ WG3 N 498

PRODUCT DATA REPRESENTATION AND EXCHANGE

Part: **Title:** AP Development Guidelines for Shipbuilding

Purpose of this document as it relates to the target document is:

_____ Primary Content

_____ Issue Discussion

_____ Alternate Proposal

_____ Partial Content

Current Status: Working Draft

ABSTRACT:
This document is to provide guidelines for the development of Application Reference Models for shipbuilding in order to assure consistency and integration among the various shipbuilding related Application Protocols.

KEYWORDS:

application protocol
shipbuilding
guidelines
product model development
ship common model
common application activity - model

Document Status/Dates (21/11/97)

Part Documents	Other SC4 Documents
______ Released	21/11/97 Released
______ Project	_______ Working
______ Working	_______ Editorial OK
______ Technically Complete	_______ Technically Complete
______ Editorially Complete	
______ ISO Committee Draft	_______ Approved

Owner/Editor: Tim Turner
Address:
LR
100 Leadenhall Street
London, EC3A 3BP
ENGLAND
Telephone/FAX: +44 171 423 2047 / 2063

E-mail:tcstjt@aie.lreg.co.uk or Tim.Turner@lr.org

Alternate: Peter Lazo
Address:
Newport News Shipbuilding
Newport News

USA
Telephone/FAX: +1804 688-8314 / -1073

E-mail: plazo@mcimail.com

Comments to Reader

The document includes the changes agreed at the STEP meeting in Chester (March 1997), and in San Diego (June 1997), such as guidance on empty select types, redeclarations, constraints, assumptions on “oneof” subtypes, BB referencing, use of STRING, user-defined options, graph principles and external_referencing. It was released during the Florence STEP meeting (19/10/97) for review by the Ship Team, although some parts are still under development & this is clearly indicated, whilst there is also some consistency to be addressed (e.g. between different diagrams).

EXECUTIVE SUMMARY

This document provides the community of people developing Application Reference Models for shipbuilding with a set of guidelines in order to assure consistency and integration among overlapping Application Reference Models, and provides a detailed description of the structure of the overall Ship Product Model, it’s architecture, design and use. In addition the guidelines aim for a more efficient development of Application Reference Models and related documentation through the concept of reuse and automatic generation of documentation.

The document defines among other things

- general modelling concepts and rules, especially the Building Block approach
- proposals for solving interoperability issues
- the formal definition and the requirements for commenting of Building Blocks
- the Ship Common Model architecture, it’s framework, domain models and utilities
- a list of quality criteria to be fulfilled by product model contributions.
- the shipbuilding Common Application Activity Model

These guidelines introduce concepts, which do *not* exist in the context of the development of Application Protocols in accordance with the ISO/STEP development guidelines. However, great care has been taken such that the result of the application of the guidelines here can be easily transformed to yield a representation of an Application Reference Model as required by the ISO/STEP guidelines.

These guidelines do not intend to supersede the ISO/STEP Application Protocol development guidelines in any way. It shall be viewed as a supportive means for the concurrent development of two or more overlapping Application Reference Models.

This document is intended to be a living document amended through practical experience. Feedback should be provided to the document editors.

This document now represents the harmonisation of several previously separate, but related pieces of work, such as the previous release of ISO TC184/SC4/WG3/N498-AP Development Guidelines for Shipbuilding, dated 17/07/96 [Guidelines-96], ISO TC 184/SC4/WG3 N511 - Shipbuilding Common Activity Model, dated 14 January 1996 [Common AAM] and the Ship Common Model, Version dated 30 July, 1997 [SCM-97].

DOCUMENT HISTORY

Version dated September 8, 1995:	Based on the MARITIME Cookbook, Version 2.0; numerous changes; editor: Roland Oehlmann, BIBA..
Version dated January 21, 1996:	Naming convention for stubs documented; EXPRESS restrictions revised (complex entities); term Ship Common Model introduced and BBs assigned.
Version dated April 18, 1996:	Graphs Building Block included; BB list for Ship Common Model updated; text on Building Block names added; reference to the AP Cross Reference List added; requirements for the BB header changed to not include UoD Classification, Activity Model Reference, and Exported Entities and Types, but Open Issues and a Rationale.
Version dated July 17, 1996:	Rule on import of abstract supertypes from BBs included (clause 3.11); discussion on interoperability added (clause 3.10); commenting of enumerations (clause 3.6.1/2); update of BBs in the Ship Common Model (clause 4.1).
Version dated October 19, 1997:	Updated rule on how to reference STEP resources (clause4.6); added rule on use of enumerations (clause 4.5.2), use of complex_entities (clause 4.5.3), and use of STRING (clause 4.5.4); updated rules on select types (clause 4.5.5), redeclared attributes (clause 4.5.6), instance level references (clause 4.9), added guidance on when external references should be used (clause 4.9.1); added section on design of external references (clause 4.9-4.9.1 & Appendix B), plus example usage of external references (Appendix B); Added section on compilation of building blocks (clause 4.11); and Added the principles behind the graphs building block (clause 5.6.10). Merged Guidelines with the Ship Common Model. Introduced EXPRESS-G for most Building Blocks within the SCM. Added Common Application Activity Model, chapters on Ship General Characteristics and Location concepts, removal of Classification Survey & Moulded Forms, added Ship_point, curve, surface etc. Introduction to the Building Block approach added.
Version dated November 21, 1997:	Modified front cover to include new "N" number & description in the footer.

CONTENTS

1. INTRODUCTION TO THE ISO 10303 SHIP APPLICATION PROTOCOLS

ISO 10303 is an International Standard for the computer-interpretable representation and exchange of product data. The objective is to provide a neutral mechanism capable of describing product data throughout the life cycle of a product, independent from any particular system. The nature of this description makes it suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases and archiving.

This International Standard is organised as a series of parts, each published separately. The parts of ISO 10303 fall into one of the following series: description methods, integrated resources, application interpreted constructs, application protocols, abstract test suites, implementation methods, and conformance testing. The series are described in ISO 10303-1.

The series of shipping industry application protocols assumes that the ship product model can be divided into separate ship systems that each cover a key element of the ship for its whole life cycle.

The reason for doing this is as much to do with distribution of modelling work as it is with the need to exchange subsets of the product model between agents in the Marine industry, let alone the practical aspects of exchanging the data associated with an entire ship.

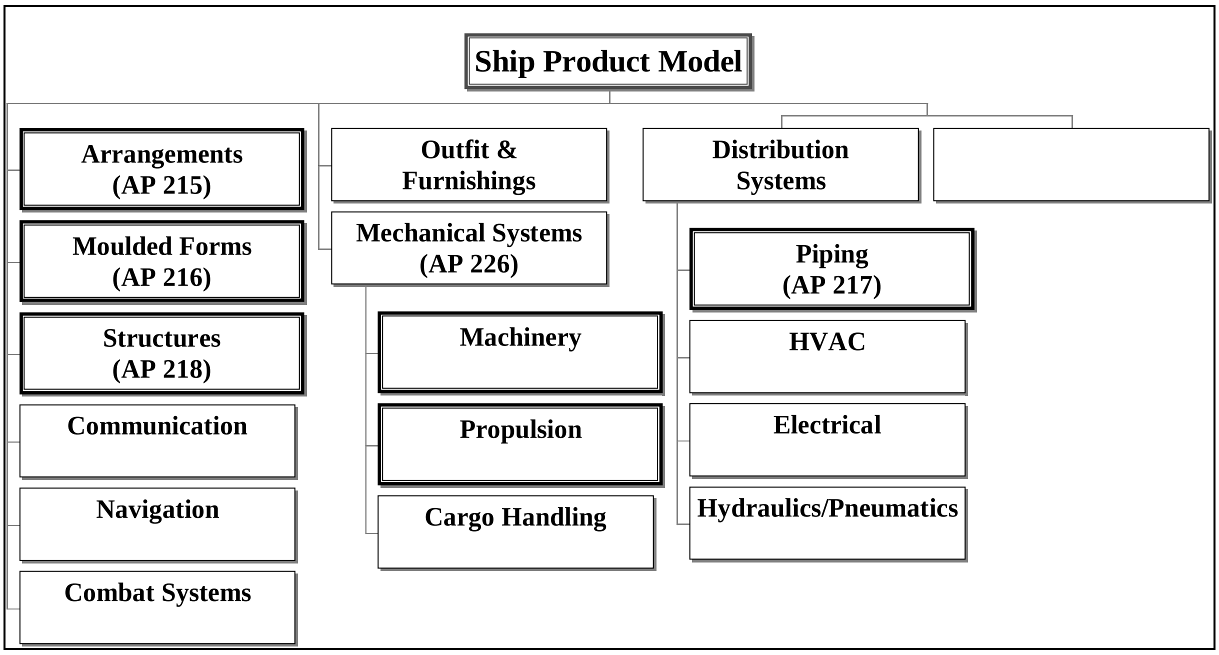


Figure 1: Ship Product Model

These key elements are shown above: ship moulded forms, ship arrangements, ship distribution systems, ship structures, ship mechanical systems, ship outfit and furnishings, and ship mission systems. Each separate system is described by one or more different application protocols. The full series of shipping application protocols (AP) is shown in Figure 1. Those boxes that are in bold indicate those which are under development at the present time.

Simply put, each AP has three major parts; an Application Activity Model (AAM) to describe and decompose the activities, input and output objects, controls and modifiers; an Application Reference Model (ARM) to describe the objects required, their structure and attributes; and finally an Application Interpreted Model (AIM) to map the requirements to the types of objects understandable to other CAD systems.

However, because applications will need (for example) to be able to build up data about all the different parts and areas of a ship, (perhaps over a number of iterative exchanges over time), not restricted to just a single AP, the data (& therefore, the ARMs) representing the various parts of the ship must also be able to be integrated. This requires that there is an overall mechanism, around which, all the shipbuilding AP's can be integrated. This mechanism should be central to each of the AP's structure such that the information is structured and organised in a consistent and similar manner and is known as the Ship Common Model.

There are several by-products of this. For example, the complexity of the models should reduce whilst the understandability of the different AP's among modellers increases, allowing for both easier interoperability and integration of the overall product model. Information can also be navigated & retrieved in a similar manner, regardless of which AP is being used, through the use of such a mechanism and can help the process of interpretation and development of the AIM.

2. Scope

This mechanism has been constructed using an approach known as the Building Block approach which is described first, in the following introduction. This approach can also be used to semi-automatically generate the *documentation* of the ARM for an AP in the shipbuilding group, and to ensure this, a set of guidelines are presented in the second section to describe all the details necessary to conform to the approach. In the third section this document describes the integration mechanism known as the Ship Common Model, it's architecture, parts and use. In the appendices you will also find a description of the common Application Activity Model [Common AAM] used for shipbuilding. This is intended, like the Ship Common Model, to provide a framework or root, from which other Application Activity Models can be developed.

2.1 Abbreviations

For the purposes of this Part, the following abbreviations apply:

AAM	Application Activity Model
AIM	Application Interpreted Model
AP	Application Protocol
ARM	Application Reference Model
BB	Building Block
CAD	Computer Aided Design
CAM	Computer Aided Manufacture
EMSA	European Marine STEP Association
IMO	International Maritime Organisation
PICS	Protocol Implementation Conformance Statement
SCM	Ship Common Model
SI	Système International
SOLAS	Safety of Life at Sea
SPM	Ship Product Model
UoF	Units of Functionality

3. Introduction to the Building Block Approach

An Application Reference Model in the context of the ISO/STEP methodology is the representation of application domain specific product model requirements (see [AP Guidelines]). The shipbuilding community is developing several such reference models, each to provide the requirements for a distinct shipbuilding Application Protocol. However, these reference models overlap with respect to the concepts found in the shipbuilding domain. The guidelines here address these overlaps. The intention is to

- make such overlaps explicit through use of identical Application Reference Model subsets;
- to provide for a more efficient Application Reference Model development through the reuse of elements (potentially) common to two or more Application Reference Models.

The principle means to support this is the concept of *Building Blocks*. A Building Block is a generic EXPRESS-based construct for the confined representation of a Unit of Functionality in part or in whole. Therefore, a Unit of Functionality may be represented by a single Building Block or many.

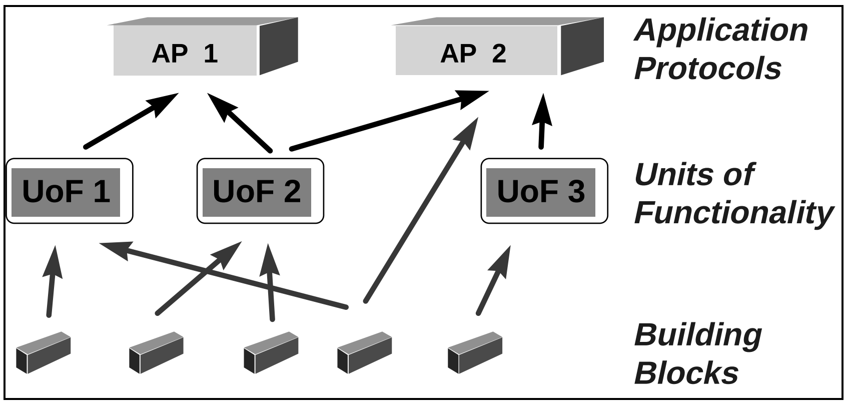


Figure 2: Building Block Approach

The Building Block concept and other elements introduced in this document do *not* exist in the context of the development of Application Protocols in accordance with the ISO/STEP methodology [AP Guidelines]. However, great care has been taken such that the result of the application of the guidelines here can be easily transformed to yield a representation of an Application Reference Model as required by the ISO/STEP methodology.

3.1 Defining Building Blocks

A *Building Block* (BB) is an EXPRESS-based specification that shall be used for the definition of Units of Functionality. A Unit of Functionality may include none, one or several Building Blocks. A Building Block consists of three schemas,

- an *import* schema providing an interface for those elements of other Building Blocks to be used by the model schema of this Building Block,
- an *export* schema, making available those elements of the model schema intended to be used by other Building Blocks, and
- a *model* schema, actually modelling the Unit of Functionality to be represented by this Building Block. A model schema may *USE* or *REFERENCE* only from the import schema.

Note: An export schema shall state only those elements, which are potential candidates for being used by other Building Blocks. Interpret this guideline rather restrictively!

In addition to the schemas, each Building Block shall have a Building Block Header.

3.1.1 Building Block Syntax – Example:

```
(*
Building Block Name:    structural_parts
Editor: Thomas Koch (KCS)
E-mail: tk@kcs.se
Version: $Revision: 1.2 $
Status: $State: Draft $
Last Edit: $Date: 1993/09/30 13:00:12 $
Description: A structural part contains the properties common to
all elements of a structural system.
Open Issues:
Rationale: The structural part BB is created based on requirements
from AP218.
*)
```

The above header allows some automatic processing by the Building Block e-mail server. A template for such a header can be obtained from the Building Block e-mail server (see section 7).

Note: the fields in the header (denoted by "$<field-name>: <contents> $") for example, "$Date: 1993/09/30 13:00:12 $", must be maintained if the processing is to succeed. However, the <contents> field may be left empty as this will be replaced by the email server when the schema is checked in with the relevant information.

```
SCHEMA example_import;
    USE FROM ship_parts_export (ship_part);
    USE FROM global_reference_system (location_on_mould_line);
END_SCHEMA;
```

The import schema describes all necessary links to other Building Blocks

```
SCHEMA example_export;
    USE FROM example_model (structural_part);
END_SCHEMA;
```

The export schema makes those elements of the model schema (below) public which may then be used by other Building Blocks[1].

```
SCHEMA example_model;
    ENTITY structural_part
      ABSTRACT SUPERTYPE
      SUBTYPE OF (ship_part);
        location: location_on_mould_line;
     (*...*)
    END_ENTITY; (* structural_part *)
END_SCHEMA;



(*
changes from previous versions:
- ...
*)
```

Following the Building Block body, that is after the final END_SCHEMA statement, changes to the Building Block compared to previous versions shall be stated in an informative, but detailed way.

Please note, that the name of the schemas of a Building Block shall be constructed as in the above example, adding the terms *import*, *export* and *model* to the BB name to indicate the schema type.

[1] However, see the section on BB usage

For further information on the Building Block methodology see [N327].

4. MODELLING GUIDELINES

4.1 Building Block Name

The name of a Building Block shall be easy to read and understand; abbreviations should not be used. To avoid confusion of names the BB name is recommended to be plural. Thus, the main construct that is covered by the BB can get the same name in singular.

Example: Building Block definitions - ENTITY Definition;

4.2 Size of Building Blocks

To give a concrete size of a Building Block in terms of lines or pages may not really be helpful[2]. However, the following guidelines indicate, whether the size is reasonable:

- too large: is the scope such that the Building Block cannot be reused without tying in unneeded Units of Functionality? E.g., the combination of Wireframe and B-Rep geometry into a single Building Block is inappropriate.
- too small: is the conceptual scope such that the Building Block represents only a portion of one Unit of Functionality and is not reused in another UoF? E.g., the split of the approval concepts into two Building Blocks containing approval status and approval item respectively is inappropriate.

4.3 Existing Building Blocks

A list of existing Building Blocks is maintained in the AP Cross Reference List [LR12ADP] of the STEP AEC Shipbuilding group. There both BB name, last update, owner and using APs are noted.

4.4 Commenting Building Blocks

The following provides guidelines on how to comment a Building Block. Please read this section carefully, there are many guidance rules embodied within this section. The reasons for having these guidelines are;

- to ensure that comments appear where necessary;
- to be able to produce STEP-related documents (semi-)automatically. Automatic processing provides the advantage that only a *single* source (the Building Block) needs to be maintained manually. Otherwise, changes of an entity or schema common to e.g. several Application Protocols would require manual changes throughout.

Note: In order to produce STEP-related documents and specifically clauses 4.2 and 4.3 of Application Protocols automatically and in high quality, the **exact conformance** to these guidelines is essential! Further information about tools for such automatic processing can be obtained from the editor (see cover page).

Note: The comment for an entity or attribute in a Building Block shall *not directly* represent an Application Object respectively an attribute definition as required for clause 4 of an AP. A comment shall only fulfil those requirements, which are sufficient to (automatically) create such definitions. These requirements are stated in this section 4.4. Nevertheless, all Building Block developers

[2]Nevertheless, a Building Block of say 5 pages or more will attract more likely a critical look then say a Building Block of 2 or 3 pages.

should be familiar with the latest version of the *Supplementary directives for the drafting and presentation of ISO 10303 (ISO TC184/SC4).*

An example illustrating all of the following guidelines can be found on page 12.

4.4.1 Where to place comments

The following EXPRESS constructs shall be commented:

- (model) SCHEMAs
- TYPEs
- the individual entries of ENUMERATIONs
- ENTITYs
- attributes (including derived and inverse attributes)
- WHERE and UNIQUE clauses
- FUNCTIONs and PROCEDUREs, and
- RULEs .

In addition, the *arguments* of FUNCTIONs, PROCEDUREs and RULEs shall be commented.

A comment should immediately follow the semicolon terminating the introductory declaration of these constructs. The only *exception* is the declaration of FUNCTION, PROCEDURE, or RULE argument lists, where the last argument declaration is terminated by ")", omitting the semicolon. In that case the comment should be inserted between the end of the declaration and the ")".

The declaration and comment shall be separated only by **any number and mix of white space** characters (e.g. blanks, tabs and newlines).

Be aware that the Building Block *description* is used as the SCHEMA comment.

4.4.2 How to format comments

A comment itself may be formatted as thought appropriate. It should be a comment enclosed by "(*" and "*)". The single line comments, each introduced by "--" are allowed by EXPRESS, and are legal syntax, but it is strongly advised not to be used as line wrapping sometimes separates the "--" from the rest of the comment creating errors when parsing the schemas.

The comment of a SCHEMA, TYPE, ENTITY, FUNCTION, PROCEDURE, RULE, INVERSE-clause, WHERE-clause, or UNIQUE-clause declaration shall start with a "proper", i.e. *complete English sentence*.

The comment of an attribute (except for an INVERSE attribute) declaration, an enumeration entry, and a FUNCTION, PROCEDURE or RULE argument shall form a *half-sentence* (noun clause). This sentence shall begin in such a way, that it makes up the *right hand part* of the following comment structure:

<identifier commented on> : <right hand side to be supplied as comment>.

Given as an example the attribute `centre: point`, the comment could read for example `the centre of the local co-ordinate system relative to its global co-ordinate system`. This would then appear as follows in a BB;

```
ENTITY  XXX;
(* Entity comments *)
```

```
  centre: point;
  (* the centre of the local co-ordinate system relative to its global co-
  ordinate system. *)
END_ENTITY;
```

Don't state complete sentences, *don't* use verbs in the beginning part of such a comment! When having finished commenting a first Building Block, please, check again conformance to this rule!

Sentences and half-sentences (see above) within comments shall be separated respectively terminated by the appropriate punctuation (colon, semicolon or point). A comment sentence following a comment sentence shall start with an uppercase letter, unless punctuation rules require otherwise.

- No hyphenation shall be used to continue a comment on a subsequent line.
- Any comments in a line following an EXPRESS construct will *not* go into the final document.
- An empty line within a comment indicates the start of a new paragraph in the final document.
- Avoid any other formatting, it will not be reflected in the final document.

4.4.3 Keywords in comments

There are three classes of comment elements, which shall be highlighted in certain way. These are:

- *Examples*:

  An example in a comment shall be enclosed by the keywords "**EXAMPLE:**" and "**END_EXAMPLE**" written in uppercase letters.

- *Notes*:

  A note in a comment used to emphasise a certain issue shall be enclosed by the keywords "**NOTE:**" and "**END_NOTE**" written in uppercase letters.

- *References to identifiers*:

  A reference to an identifier stated in the EXPRESS code of the current Building Block shall have the form "<"identifier">". If an identifier is used in the plural form within the comment but stated in the singular from in the EXPRESS code, the reference should be as follows: "<"identifier">s". Identifiers shall appear exactly as they have been stated, i.e. including underscores, if used, and with the same uppercase and lowercase letters. To avoid typing errors, it is recommended to copy and paste long identifiers into comments.

- *Illustrations*:

  Authors who would like to illustrate their Building Blocks may reference a PostScript file (encapsulated postscript) from the Building Block. The reference should contain the following information: a tag "FIG" , the authors initials and a file name. Example: < **FIG** AN-pic_5 >. The authors have to take care that the files are sent to the Building Block server.

Consequently, the above keywords and symbols **shall not** appear within a comment except for playing the above roles.

4.4.4 Example

```
SCHEMA version_model;
   (* This schema provides a general versioning mechanism. It
      supports concepts of single version derivation, alternatives
      and merging of versions. It also supports the concept of
      "current" versions.
   *)
```

```
USE FROM version_import (identifier, label, node,
   directed_open_arc, directed_acyclic_graph);

ENTITY version
   ABSTRACT SUPERTYPE
   SUBTYPE OF (node);
      (* a <version> is an element of a <version_history>
         described as a version graph. It identifies a
         piece of work within the set of alternatives and
         direct and indirect predecessors and successors of
         this piece of work in the <version_history>.
         NOTE: This schema does not foresee, nor define, what an
            item to be versioned actually is. A proposed
            mechanism to incorporate the concept of versioning
            is to inherit he appropriate versioning constructs.
         END_NOTE
      *)
   context: version_history;
      (* the version graph the <version> belongs to. *)
   name: identifier;
      (* the identifying name of the <version> within the set
         of <version>s defined by the <version_history>.
         NOTE: the <name> of a <version> is not intended to
            indicate the position of a <version> in a
            <version_history>. This structure is represented
            explicitly through the association of two <version>s
            modelled as a <version_relationship>.
         END_NOTE
         EXAMPLE: often encountered forms of names for software
            versions are "beta release", "0.1", "2.a" etc.
         END_EXAMPLE
     *)
INVERSE
   current_declaration: SET OF current_version FOR current;
      (* a <version> may be defined zero or more times as
       current. *)
UNIQUE
   contxt, name;
      (* the name of a <version> together with its context
       shall be unique within the its <version_history>. *)
WHERE
   defined_in_context: version_in_context(SELF, contxt);
      (* a <version> shall be part of a <version_history>. *)
END_ENTITY;

FUNCTION version_has_context
   (
      vrs: version;
         (* the <version> to be checked whether it is part of a
          certain context. *)
      contxt: version_history
         (* the context the <version> shall be part of. *)
   ): BOOLEAN;
      (* the <version_has_context> function checks whether a
         <version> is part of a certain <version_history> or
         not. A <version_history> is referred to here as a
         context.
      *)
   RETURN (vrs IN contxt.nodes);
END_FUNCTION;

...
```

4.4.5 Identifiers

Names of entities and entity attributes shall be *nouns* or *qualified nouns*. They shall *not* be verbs, abbreviations etc. For example, do not use names like `lib` (for library), `for_production_part`, `defined_by` etc. There are a small number of exceptions to this rule (e.g. defined_for) within the Ship Common Model, but these have been reached through agreement within the group.

Note: this restriction is in addition to the above commenting rules necessary for automatic document creation.

An entity name shall start with an upper case letter followed by lower case letters and underscores as necessary.

An attribute name shall start with a lower case letter followed by lower case letters and underscores as necessary.

4.4.6 Contents of Comments

In addition to the formatting guidelines stated above, this section provides guidance on what makes a good definition.

Each SCHEMA comment shall include a statement of the content of the schema using Application area terminology (e.g., an experienced structural engineer, naval architect, or designer).

Be aware that entity comments will become the STEP Clause 4 Application Object and attribute definitions. They should also be written using application terminology. Each definition should describe the "real world" concept that an entity represents, and should not restate the EXPRESS definition. The following sections provide some guidelines for writing definitions and questions that Building Block developers should ask themselves to critique their definitions.

Entity comments; The following rules apply to entity comments:

- The comment shall consist of a description of the entity defined within the context of the AP(s).
- The fact, that an entity is a sub- or supertype of some other entity shall *not*[3] be stated in the comment.
- The attributes associated with the entity shall *not* be enumerated in the comment (see footnote 3).

Entity attributes: The following rules apply to the documentation of entity attributes:

- The text that follows the name of the attribute shall consist of a description of the attribute defined within the context of its entity.
- In the case of a logical attribute, the terms "logical flag" or "logical indicator" shall not be used; both conditions that the attribute creates shall be indicated.
- The fact that an attribute is optional shall *not* be stated in the comment (see footnote 3).
- The fact, that an attribute is an aggregation shall *not* be stated in the comment (see footnote 3).
- The individual, possible values of an attribute of an enumeration type shall *not* be stated in the comment (see footnote 3).

4.4.6.1 Criteria for Lexical Definitions

The following recommended practices are quoted from *A Concise Introduction to Logic* by Patrick J. Hurley**:**

"Because the function of a lexical definition is to report the way a word is actually used in a language, lexical definitions are the ones we most frequently encounter and are what most people mean when they speak of the 'definition' of a word. Accordingly, it is appropriate that we have a set of rules that we may use in constructing lexical definitions of our own and in evaluating the lexical definitions of others."

Rule l: A lexical definition should conform to the standards of proper grammar.

Rule 2: A lexical definition should convey the essential meaning of the word being defined.

The attributes mentioned in the definition should be the important or necessary features of the thing defined, not trivial ones.

[3]This information is redundant and can be created automatically from the EXPRESS code for clause 4 of an AP.

Rule 3: A lexical definition should be neither too broad nor too narrow.

A definition is too broad if the definition applies to things other than the things that are being defined.

Rule 4: A lexical definition must not be circular.

A circular definition uses the definiendum in some way in the definition and is thus not genuinely informative.

Rule 5: A lexical definition should not be negative when it can be affirmative.

Rule 6: A lexical definition should not be expressed in figurative, obscure, vague, or ambiguous language.

A definition is figurative if it involves metaphors or tends to paint a picture instead of exposing the essential meaning of a term. Example: "Architecture" means frozen music. A definition is obscure if its meaning is hidden. One source of obscurity is overly technical language. A definition is vague if its meaning is blurred. A definition is ambiguous if it lends itself to more than one distinct interpretation.

Rule 7: A lexical definition should avoid affective terminology.

Affective terminology is any kind of word usage that plays upon the emotions of the reader or listener.

Rule 8: A lexical definition should indicate the context to which the definition pertains.

This rule applies to any definition in which the context of the definition if important to the meaning of the definiendum. Whenever the definiendum is a word that means different things in different context, a reference to the context is important.

4.4.6.2 Self-assessment questions for Entity or Application object definitions

The following questions should be used for the evaluation of entity or application object definitions. They are drawn from the Qualification Manuals.

1. Is the definition understandable as written? Does it present the meaning of the object in a clear and succinct manner?
2. Do the EXPRESS language constructs correspond to the textual definition?
3. Is the name of the entity appropriate? Does the attribute name make sense when read with its entity? Does the name correspond to the definition?
4. Does the definition adhere to the rules of grammar and make good use of English?
5. Are there any unusual or domain-specific terms used that need clarification, replacement or definition?
6. Is an illustration, example or explanatory note necessary to understand the definition? (or is such an illustration or example available to aid the understanding?)
7. Does the definition/entity adhere to the scope?
8. Are references to informative material necessary?
9. Is the entity/object necessary within the context of its definition (e.g., the particular clause, schema or application object in which it is defined)?
10. Is the object like some other object? Can the definition be merged with some other object or can another object be used in its place?

11. Is this a commonly accepted term within the application domain? If the definition is the same at the one in the Oxford English dictionary, it does not need to be defined. Is the usage of the term within STEP different from normal or accepted application domain usages? (If so, pay attention to the clarity and sufficiency of the definition.

4.5 Restrictions on Usage of EXPRESS

4.5.1 Use of OPTIONAL Attributes

There is a potential danger of misusing the concept of optional attributes. This becomes evident when a model has a relatively "flat" structure with many optional attributes. In many cases this can be overcome through the use of sub-super typing.

Typical examples for misuse are:

- Mix of concepts in a single entity. To have meaningful instances of such an entity, only a subset of the attributes shall be initialised for an instance. Consequently certain attributes have to be rendered OPTIONAL. This situation can be avoided by subtyping. The common attributes define a supertype, while the special attributes are aggregated in two or more distinct subtypes.
- Uncertainty about the meaning and relevance of an attribute. In that case the attribute shall not be included until the open questions are resolved.

Use optional attributes restrictively and with care!.

4.5.2 Use of Enumerations

Enumerated types are frequently used. However, where there is a doubt about the list that has been enumerated, a "user-defined" option can be used. For example, suppose we have an entity named "panel", one of the attributes of panel might require the type of panel to be made explicit. This might be done using standard_panel_type, which is an enumerated type listing all the known types of panels as shown below.

```
ENTITY panel
      type_of: standard_panel_type;
    .
    .
END_ENTITY; (* panel *)

TYPE standard_panel_type = ENUMERATION OF
    (side,
     bilge,
     transverse_wash_bulkhead,
     aft_bulkhead,
     strength_deck);
  END_TYPE; (* standard_panel_type *)
```

However, if there is a doubt that the enumerated list might be incomplete, or that special cases need to be catered for, then we can introduce a "user_defined" option as shown below. Effectively, this is used in conjunction with an additional optional attribute used to describe the case that is not catered for in the list. Where the enumerated type list does cater for the case of concern, then the optional attribute should not be used (during instantiation).

```
ENTITY panel
      type_of: standard_panel_type;
```

```
         user_defined: OPTIONAL TEXT;
       .
         .
   END_ENTITY; (* panel *)

   TYPE standard_panel_type = ENUMERATION OF
       (side,
        bilge,
        transverse_wash_bulkhead,
       user_defined,
       aft_bulkhead,
        strength_deck);
     END_TYPE; (* standard_panel_type *)
```

4.5.3 Use of ANDOR

ANDOR constructs shall be used with care to avoid a proliferation of complex entities. In some cases, however, they are very efficient modelling constructs (see also section 5.4.4)

Care should be exercised when defining super/sub types, so that they do not span Building Block boundaries. For example, when a new super type is introduced into a Building Block, it's sub-types should be created within that schema whenever possible. Exceptions to this rule include those Building Blocks of the Ship Common Model which forms the framework for the Application Protocols and are meant to be generic super types.

4.5.4 Use of STRING

All string attributes should be used in a consistent manner. Part 41 provides specific STRING types which carry specific semantics. However, there are two human interpretable string types: LABEL and TEXT. The type ID is an identifier which is not necessarily human interpretable. These types should be used accordingly. Where a specific STRING (or set of) are expected, then these should be made explicit through the use of an ENUMERATION type.

4.5.5 Use of SELECT Types

When SELECT types are used with care they can be extremely useful. SELECT types should be used where the types defined differ in their own whilst they play the same role in another. Therefore, a SELECT type can be used to choose between a number of different representations of a particular concept, which might differ depending upon the context. For example, "time" can be represented as either a <date>, <local_time> or both (date_and_time). In some circumstances, only a date would be available (or needed) whereas at others the local time would be required. The example below shows how a SELECT type can be used to provide for this.

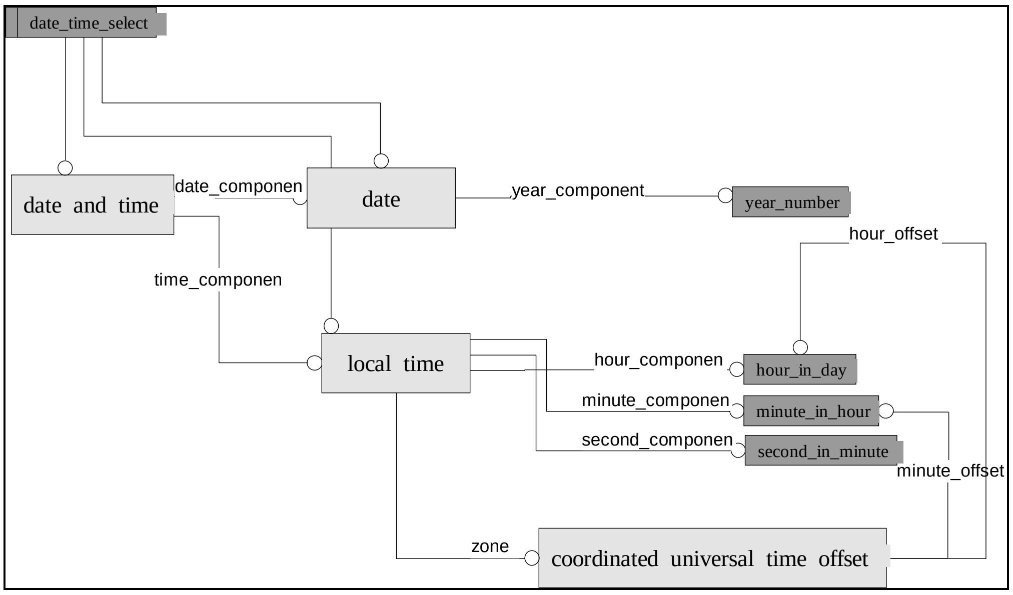


Figure 3: Example SELECT type

Otherwise a SELECT type can be used to allow the user to specify which level of detail should be provided. For example, imagine a set of sub-super type relations where more and more attributes are provided the more specific the sub-type. During an iterative exchange of information, the information may become richer as more and more information is exchanged. The level of information can be specified through the use of a SELECT type referencing sub-types at various levels of granularity, allowing more & more of the attributes to become available during subsequent transfers.

From a compilation viewpoint, each entity referenced from within a select statement that is not contained within the current schema, should be explicitly listed in the import schema for that BB. For example, in the external_references BB the following `SELECT` type is defined;

```
TYPE Any_address = SELECT (Address,Universal_resource_locator);
    (* Either an ordinary postal address (Address) or a computer ad-
  dress(Universal_resource_locator) *)
END_TYPE;
```

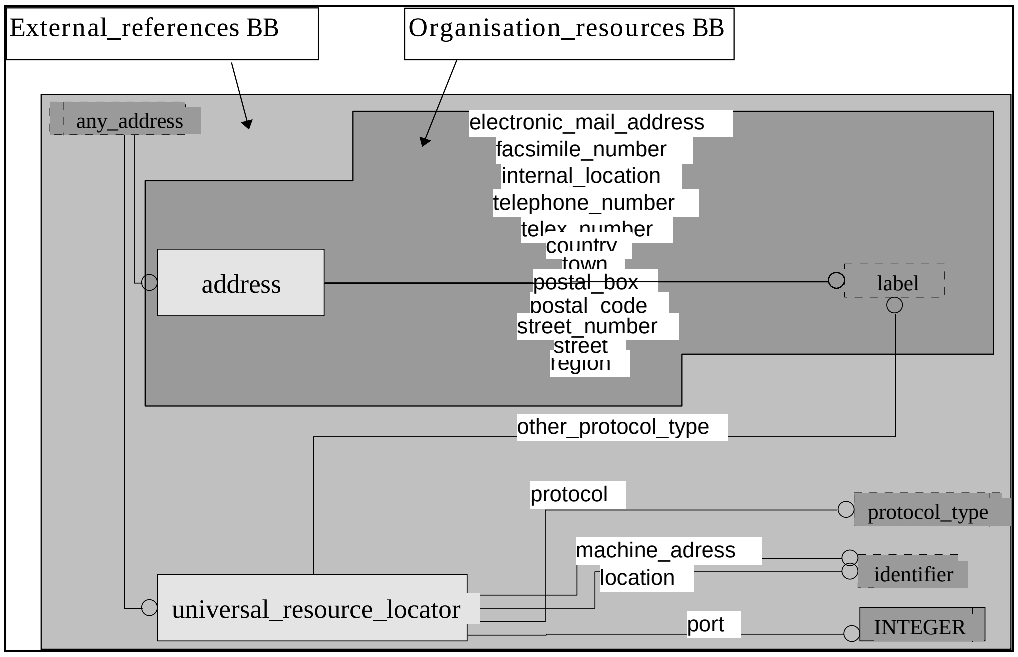


Figure 4: SELECT types across schemas

Therefore, when using SELECT types such as <any_address> from within a schema which imports this type from the external_references BBs, <address> is defined only indirectly and will need to be explicitly imported. This is because EXPRESS prunes out entities referred to in “SELECT” statements that are outside the current schema and not explicitly imported. This might appear a little odd since the external_references BB itself imports the BB where Address is found and it is only natural to assume[4] that this would be inherited some how. Therefore, the following statement should appear in the schema where <any_address> will be used.

```
SCHEMA some_schema_import;
     USE FROM organisation_resources_export (Address);
END_SCHEMA;
```

See Figure 4: SELECT types across schemas above and also Figure 5: Global_id & External References within the SCM.

4.5.6 Use of Redeclarations

A redeclaration shall reference either the entity in which the attribute was originally declared or an entity in which it has been redeclared already. Consider the following;

```
SCHEMA retype_test;

ENTITY Original;
  org_attr: NUMBER;
END_ENTITY;

ENTITY Middle
  SUBTYPE OF (Original);
END_ENTITY;

ENTITY Low
  SUBTYPE OF (Middle);
  SELF\Original.org_attr: REAL;
END_ENTITY;

END_SCHEMA;
```

A statement: SELF\middle.org_attr: REAL; in entity Low would be wrong in the context above. See also section **Error! Reference source not found.** for usage guidance.

In addition it is also necessary to import explicitly the entity with the original attribute (entity Original above) into the current schema with the redeclaration; implicit reference is not enough. This entity needs to be taken into the "name space" of the current schema. This is done by including the entity into a USE FROM list such as;   USE FROM <Building block> (<Entity>);

For example, consider the following;

```
SCHEMA A;
  ENTITY A_1;
     an_attr: NUMBER;
  END_ENTITY;
END_SCHEMA;


SCHEMA B;
```

[4] EXPRESS denies this as part of it’s pruning strategy

```
    USE FROM A(A_1);
    ENTITY B_1
       SUBTYPE OF(A_1);
    END_ENTITY;
  END_SCHEMA;

  a)                         or b)                  or c)

  SCHEMA C;                  SCHEMA C;              SCHEMA C;

  USE FROM B(B_1);           USE FROM A(A_1);        USE FROM B(B_1);
                             USE FROM B(B_1);

  ENTITY C_1                ENTITY C_1              ENTITY C_1
   SUBTYPE OF (B_1);         SUBTYPE OF (B_1);       SUBTYPE OF (B_1);
  SELF\B_1.an_attr:REAL;     SELF\A_1.an_attr:REAL;  SELF\A_1.an_attr:REAL;
  END_ENTITY;                END_ENTITY;            END_ENTITY;
END_SCHEMA;                 END_SCHEMA;              END_SCHEMA;
```

Definition a) is not correct because due to the group reference (\) B_1 does not see its inherited attribute "an_attr". See the minutes (produced 14-Mar-97) of the Hull Cross Section Validation Workshop at Lloyd's Register held February 1997 for some logic on the visibility issue:

Definition b) is the only solution that works.

Definition c) is not correct because A_1 is not visible in schema C, i.e. is not within its name space. The SELECT type cannot be used to "expand" a domain according to P11 (chapter 11.4.2).

4.6 How to Reference STEP Resources

This section describes how to define Application Objects and the EXPRESS representation of *Non-Shipbuilding specific* Units of functionality (NSU) as part of an Application Reference Model. Examples of such units are: date_time_resources, geometry_model_resources, geometry_resources, organisation_resources, representation_resources, support_resources, topology_resources and potentially others.

Previously, the group had created "dummy" definitions to mirror the STEP resources, but these were insufficient when compiling and validating the models using them. A decision was therefore taken to use the BB three schema approach to allow those entities selected (via the import schema) to be referenced as fully attributed definitions.

Whilst useful from the implementation point of view, when creating the ARM of an AP document, the entities within the resources should only be identified by the entity name and the originating (STEP) schema (i.e. not to be explicitly defined in the documentation) to avoid confusion during the interpretation process when the mapping of the STEP resources will be redundant.

The basic assumptions are:

- whilst necessary, the Shipbuilding Group will maintain an interface to the STEP Resource schemas through the use of the Building Block approach mentioned above (previously termed "stubs");
- the name of a NSU (or stub) shall end with the name "_resources" to allow easy identification;
- an NSU shall import the relevant entities from the STEP resources, USE them in the model schema, before exporting them in the export schema.
- the whole NSU should be explicitly referenced as an application object whilst the EXPRESS/EXPRESS-G definitions in clause 4 and Annex G of an application protocol, may present more details of the NSU if desired.

- the interpretation of an NSU will be driven by a presumed one-to-one mapping between a NSU and STEP generic resources.

For implementers, this means that the interfaced & defined entities can then be referenced by other schemas and enable correct parsing of a closed set of schemas.

A reference to a (STEP) resource and its constructs is done like the reference to any other EXPRESS schema:

```
SCHEMA hydrostatics; (* just an example *)
  USE FROM geometry_resources_export (Cartesian_point, ...);
  ...
  ENTITY Hydrostatic_position_value;
    location: Cartesian_point;
  END_ENTITY;
  ...
END_SCHEMA;
```

Summing up, traces of an NSU would be visible as part of the AP as a schema definition in clause 4 as application objects without attributes and as part of application assertions.

It is strongly suggested that stub “owners“ are also responsible for maintaining a corresponding full version for implementation purposes.

The following is an example of a stub being complete in terms of its role as a “_resource” Building Block:

```
(*
Building Block Name:  date_time_resources
...
*)

SCHEMA date_time_resources_import;
(* The schema <date_time_schema> is defined in Part 41
   *)
 USE FROM date_time_schema (date,
                calendar_Date,
                ordinal_date,
                week_of_year_and_day_date,
                local_time,
                date_and_time);
END_SCHEMA;

SCHEMA date_time_resources_export;
   USE FROM date_time_resources_model ( Date, Local_time, Date_and_time,
      Calendar_date, Ordinal_date, Week_of_year_and_day_date);
END_SCHEMA;

SCHEMA date_time_resources_model;
   (* The schema <date_time_resources> provides concepts to specify
      date and time.
   *)

   USE FROM date_time_resources_import;

END_SCHEMA;



(*
changes from previous versions:
- ...
*)
```

4.7 How to Express Cardinality Constraints Across Building Blocks

One of the main purposes of the Building Block approach, the reuse of individual Units of Functionality and corresponding modelling constructs, does not allow arbitrary *mutual* references between two or more Building Blocks. Reusing a specific Building Block would otherwise impose the inclusion of other, potentially unwanted Building Blocks.

As a consequence, cardinality constraints by means of the INVERSE clause of EXPRESS cannot be stated as if in a single schema, if the relationship between the two entities spans two Building Blocks.

Cardinality constraints shall be expressed having the following guidelines for relationships between Building Blocks in mind:

- A constraint may be stated as usual by an INVERSE clause, if the two Building Blocks together represent a single Unit of Functionality, where one of the two Building Blocks is never expected to be used just on its own. **Note**: use this rule with care.
- The inverse cardinality (0:?) exists always *implicitly* and by default, unless some specific constraint is given.
- The inverse cardinality (0:1) can be modelled using a unique clause:

  ```
  ENTITY a;
    definition: b;
    ...
  UNIQUE
    UR1: definition;
  END_ENTITY;
  ```

  The unique clause requires, that `b` is related to none or one `a;`
- In order to express other cardinality constraints, a where-rule is required.

4.8 How to Reference Functions

A function declared in a foreign schema can be visible in another schema only via a REFERENCE interface. This referenced foreign function cannot be referenced by other schemas again, because a referenced element is per EXPRESS definition not treated as if it was a locally declared element. Consequently it is impossible to establish a referenced function visibility from an export schema to other import schemas in our export/import model.

A solution is to redeclare the functions in both export and import schemas. Every redeclared function will call the corresponding referenced foreign function and return its result.

Example:

```
SCHEMA s1_model;
   (* ...*)
     FUNCTION f (p1:INTEGER; p2:INTEGER):BOOLEAN;
      (* this function f should be referenced in schema s2_model.
       *)
      RETURN (TRUE);
     END_FUNCTION;
   (*... *)
END_SCHEMA;

SCHEMA s1_export;
   REFERENCE FROM s1_model (f);

   FUNCTION f_e (p1:INTEGER; p2:INTEGER):BOOLEAN;
      (* declare a new function f_e which calls the referenced
       function f and returns its result. *)
      RETURN (f (p1, p2));
    END_FUNCTION;
END_SCHEMA;

SCHEMA s2_import;
   REFERENCE FROM s1_export (f_e);

   FUNCTION f (p1:INTEGER; p2:INTEGER):BOOLEAN;
      (* declare a new function f which calls function f_e and returns
       its result. The function f has the same functionality as
       the function f in schema s1_model. *)
      RETURN (f_e (p1, p2));
  END_FUNCTION;
END_SCHEMA;
```

```
SCHEMA s2_model;
  REFERENCE FROM s2_import (f);
  (*... *)
END_SCHEMA;
```

4.9 How to Reference Instances Across Application Protocols

There are several issues related to the interoperability of APs, among others the capability of APs to reference bits of the models of other APs. A piping AP may for example want to reference a plate that is penetrated by a pipe although the plate is specified in a different AP. There are several solutions for such references to external specifications. Usually bits of a model that are required by several APs are included in all the APs that want to use them. This is the AIC approach. The shipbuilding APs also apply a second mechanism which in addition allows for referencing instances that are not within the scope of the current instance model; this is the external instance reference.

With this latter solution schema overlaps of a granularity down to single entity types can be handled; this would be impractical using AICs. Also, such overlaps could be specified at "run-time", whereas AICs require standardisation. However, in contrary to using the AIC approach an instance model does not know any of the properties or attributes of an externally referenced instance. The solution has been to introduce these attributes within key locations in the SCM framework as described in the Utilities section of the SCM.

Only entity types which carry version information should be candidates for instance references. This includes all subtypes of "Definition", and "Definable_object" (to capture item_structures and item_relationships).

4.9.1 The Need for External Instance Referencing

External instance references are required for Shipbuilding APs for a number of reasons. For example, when needing to;

1 Preserve relationships between instances across different ISO APs.
2 Preserve relationships between instances in the same ISO AP during partial transfers.
3 Preserve the identity of instances which are transferred in multiple exchanges.
4 Preserve the identity of entities during joint engineering tasks.
5 Preserve the identity of entities during concurrent engineering tasks.

Each of these are elaborated a little further below to provide guidance as to when an external reference may or may not be required.

1. Is it necessary to preserve relationships between instances which are transferred using different ISO APs. For example, a structural penetration defined in AP 218 may reference the pipe defined in AP 217 which passes through the penetration. A plate defined in AP 218 may reference a molded hull form defined in AP 216 defining a hull or bulkhead.
2. Is it necessary to preserve relationships between instances which are transferred using the same ISO AP, but in separate transfers. It is not feasible or practical to exchange an entire ship in a single transfer. Partial transfers of a ship based on systems, assembly, or spaces must be supported. In this case relationships between instances in the same AP must be preserved. For example, a pipe in one data transfer may be connected to a pipe which was transferred in another exchange.
3. Is it necessary to preserve the identity of instances which are transferred in multiple exchanges so that the instances can be intelligently processed (rather than simply duplicated). This can occur if transfers are based on several different criteria, e.g. space or system, or that several transfers of the same data are made over time. It is also necessary that the identity is based on a particular version of an instance, e.g. a "Definition" in the Ship Common Model. In addition, it may be necessary to include the instance identifier in version history so that updates can be made.
4. Is it necessary to preserve the identity of entities in order to support joint engineering tasks. For example, a contractor may exchange a design with a subcontractor who details the design and transfers it back to the contractor. The detailed design will reference instances which were transferred in the original design. In order to use the detailed design, the contractor must maintain the identity of the original design objects and also recognize any revisions to these objects.

5. Is it necessary to preserve the identity of entities in order to support concurrent engineering tasks. In a concurrent design task, identifiers are required for configuration management. Clearly, instance references must be used for this purpose. However, identifiers for additional, lower level instances may also be required, e.g. underlying geometry. Here, the identifiers are immutable and persistent in the software systems.

As a guide, at least the first two of the requirements stated above may require instance referencing, whereas the last three may be satisfied by the use of a global unique identifier (GUID) - described below.

Version information also is required in order to maintain consistency between transfers and environments. For example, a plate may reference a moulded form, but requires only the underlying surface geometry. If a reference to the moulded form is used then the referencing AP must use the moulded form schema to find the surface, i.e. the moulded form schema must be loaded and accessed. However, a direct reference to the surface would cause problems if the moulded form definition later referred to a different surface.

A global unique identifier (Global_id) identifies a unique version of a definition. Hence, a Global_id must be assigned when a new version of a definition is created. This permits concurrent updates to the same definition at several sites. This is described in more detail within section 5.6.9.2 of the SCM described in the next part of this document.

However, before concrete guidance can be provided there are still some issues[5] under discussion that will need to be reflected in this document.

4.10 Importing Building Blocks into Application Protocols

4.10.1 Abstract Entities

Abstract supertypes that are imported into an AP by a Building Block, but that are not specialised in that AP, shall be removed from the AP.

4.10.2 BB Granularity

Given that the scope of some Application Protocols is not directly identical to those Building Blocks which provide the functionality required, there may be occasions where the whole of a Building Block is not needed.

Consider the following case (raised as an issue);

BB performance: USE FROM planned_maintenances (Maintenance_schedule);

BB moulded_form_representation: USE FROM offset_table_representations (Offset_table_representation);

These USE FROM statements can make a large extension for a complete AP-model, which is compounded by the fact that AP218 does not need the ENTITY <Maintenance_schedule> and <Offset_table_representation>, but has to put them into the complete AP-model because they are imported by a BB which is used.

This could indicate several things;

[5] (However, for a summary of the issues relating to this, see the minutes from the ISO Shipbuilding meeting held in San Diego, 1997, and also Annex G covering this topic).

- that the scope of the BB’s is not fine or narrow enough, & therefore, perhaps the BB is not granular enough & should be split into several BB’s to cover this functionality.
- that the scope of the AP has been set too wide.
- that the interfaces should be re-worked.

In this instance, the resolution agreed resulted in a check on the scope of AP218 and as a result many of the BBs referenced were removed from AP218 and subsequent interfaces were re-worked.

4.10.3 Semi Automated Document Generation (Input Required)

4.11 Compiling Building blocks

A note about compilers:

When compiling Building Blocks, especially when creating a single schema from many different ones, it has been established that compiler technology differs from one compiler to another. Also, the discovery of one error may mask many others. Conversely, one error might also introduce many others, depending upon the implementation of the compiler used. It is also evident that different compilers have different compilation strategies which can be exploited when relevant. Some compilers also go to deeper levels of compilation and checking (against Part 11) than others. However, despite the different strategies and levels of implementation of the EXPRESS language, there is still no perfect tool available, and whilst this situation remains, we must make the best use of the tools available.

It should be borne in mind that a single BB is unlikely to be entirely self contained and that compilation strategies should be aimed at generating correct long forms of schemas where many entities inside many different BBs are brought together and made available from within one schema (e.g. AP218). From this perspective, it is not always necessary to ensure that every BB can be compiled from it’s own set of direct references as the BBs referenced by it will themselves bring their own necessary BBs into the longform schema. In this sense, those definitions not accessible outside BB schema walls, but which are implicitly interfaced, are resolved when the longform is generated.

Notwithstanding this, the following types of errors can usually be resolved by checking that all of the required IMPORT/EXPORT & MODEL schemas have been defined adequately.

For example errors of the following types;

> Empty Select Types, Unresolved Attribute Domains, Subtype not Defined, Attribute Redeclaration, Implicit Interface Item not visible, Undefined Entity in Group Reference, Unresolved Qualified Reference;

- may all be resolved through the proper use (as defined in the EXPRESS Reference Manual - Part 11), of the IMPORT/EXPORT & MODEL schemas. These errors may be nested to several levels deep and result in the import/export of entities not used directly within the model schema.

As a general rule, entities should only be imported directly, and therefore, exported from one BB where the entity is defined rather than indirectly through the export schema of another. Apart from this being good practice, a “third party” BB may change it’s scope or requirements removing the need to export the entity being relied upon, thereby introducing an error in the import schema.

5. HIGH LEVEL BUILDING BLOCKS- THE SHIP COMMON MODEL

5.1 Introduction to the Ship Common Model

The Ship Common Model (SCM) is a set of Building Blocks (BB)[6] which are used in the Ship Product Model (SPM) context. The Ship Common Model provides a modelling ***framework*** for the Ship Product Model, a set of ***domain***[7] (independent & re-usable) product-structure models that are required for more than one Application Protocol (AP), as well as a set of commonly used constructs or ***utilities*** such as those used for configuration control & management concepts. The goal of the Ship Common Model is to contribute to the integration and overall consistency of the ARMs of the different ship APs. Thus, it is a means of integrating the requirements specified to a uniform conceptual model.

The Ship Common Model, described in this document provides this framework and is the basis for ongoing ship AP development within the ISO shipbuilding group.

It is aimed at AP editors responsible for the production of the various AP's outlined in Figure 1, and to the Building Block owners who model the functionality used in many of the AP's.

The ISO AP Development Guidelines [AP Guidelines] state that each AP should have a data planning model defined in terms of the major units of functionality used in the AP and should show the relationships between these and the framework. This provides an implicit requirement that each AP should conform to the framework and be consistent with the modelling techniques that have been used.

5.2 Overview of the Ship Common Model

This section introduces the different parts of the Ship Common Model, which are then described in greater detail in the subsequent sections. As described above, the Ship Common Model can be split into three parts; the overall modelling framework, a set of domain independent (& re-usable) product-structure models, and a set of common utilities such as configuration management concepts and measurement units. The structure & purpose of each of these is described below.

[6] A Building Block is an EXPRESS schema that is big enough to represent a concept and small enough to be shared by several contexts. Building Blocks are inter-linked.

[7] Domain - as in an area of discourse

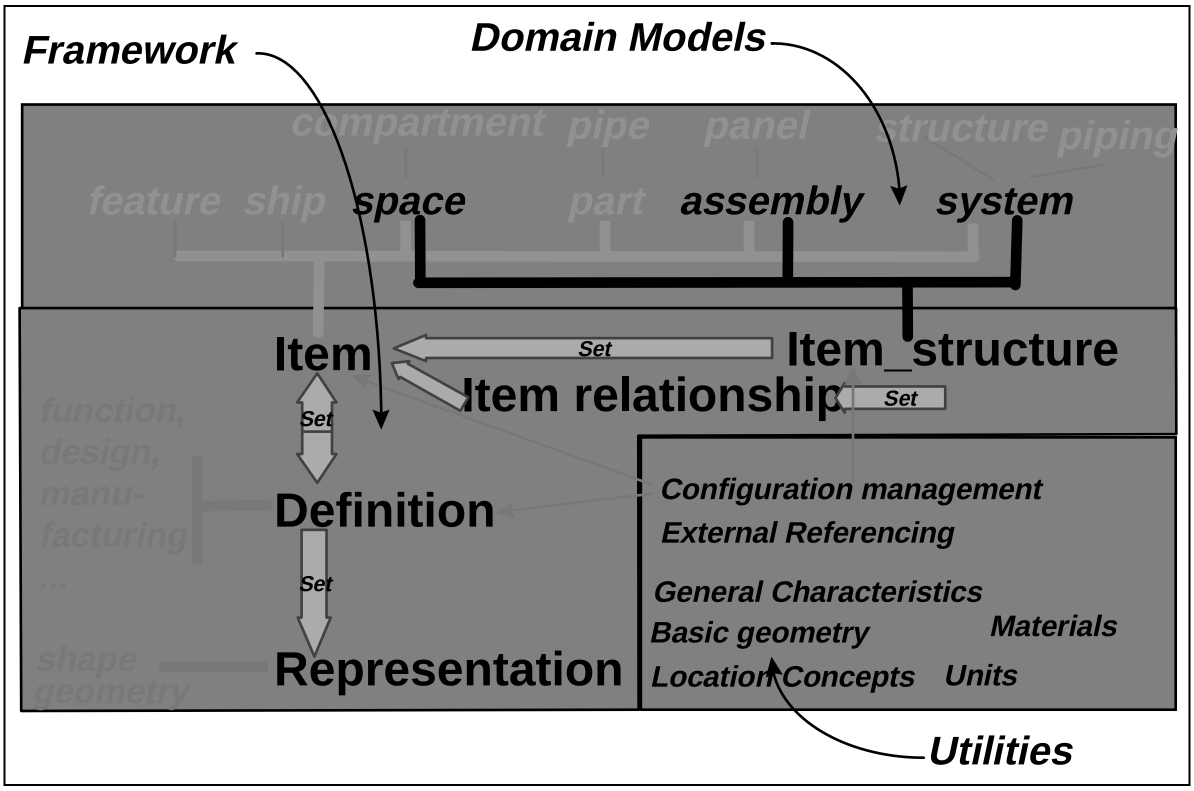


Figure 6: Parts of the SCM

5.2.1 Framework

The modelling framework part of the Ship Common Model provides the realisation of the general concepts of how to ***relate things***, how to ***define*** their ***properties*** and how to ***represent*** them. Effectively, this high level approach forces the product model to be split up across the main constructs of the framework, namely; **items**, **definitions** and **representations** whilst being linked via a number of generic relationships. One of the benefits of this approach is that it enables a better management of information such as need to organise the data according to different viewpoints and in the representation of life-cycle dependent requirements.

This framework introduces and resides in the following Building Blocks:

- definitions;
- generic_product_structures;
- representation_resources.

See section 5.4 The Framework, for a more detailed description of how the Framework is constructed and guidance on how it should be used.

5.2.2 Domain Models

On top of the framework, the domain models provide a set of templates for organising the product being modelled along a number of different axes or views, such as product by system, by space or by assembly. However, the templates also provide a set of implicit modelling techniques for the organisation of the product data. These domain models represent generic structures which, (whilst conforming to the framework) allow the modeller to organise the data of the product (through a process of specialising the generic concepts), according to their needs. In doing this the modeller negates the need to invent disparate modelling strategies for each AP product structure. The benefit of this approach is that it reduces the modelling effort, allows consistency, conformity and interoperability with the other AP's that already conform to this approach.

Some of the drawbacks of this approach are that without a proper understanding of the framework or the domain models, the structure of the resulting model may appear incongruous to the organisation first proposed by the modeller. The reasons for organising the data model along the lines of the templates are not focused entirely upon the need to reflect real world objects, but also from the needs of the framework to support interoperability, integration and consistency.

It is not intended that each of the domain models is used by a new AP. The type of structuring needed for the product model should become evident through the initial modelling work, although it is not uncommon for a number of structuring techniques to be used in an AP, and the generic domain model specialised accordingly. See section 5.5 for more detailed description on each and guidance on their use.

The current domain models[8] reside in the following Building Blocks:

- product_structure_by_system;
- product_structure_by_assembly;
- product_structure_by_space[9];
- connection_topologies[9] (product structure by connectivity);

5.2.3 Common Utilities

The Common Utilities are a group of constructs that will be required by most AP's. The utilities differ from the *Framework & Domain Models* through the fact that for the majority of cases, the utilities are ready for use and do not require any further specialisation for use in an ARM. Many have been created specifically for shipbuilding although some may be able to be used externally. The utilities group together the following Units of Functionality and their respective Building Blocks. The actual concepts used from each BB is described in section 5.6;

- ship's general characteristics; including the basic ship's length, breadth, type and class (see Building Blocks designation_characteristics and dimension_characteristics).
- product's location in relation to the ship's; including for example, the ship's co-ordinate system, local co-ordinate systems, spacing grids etc., (see Building Blocks global_axis_characteristics, local_coordinate_systems, local_coordinate_systems_with_station_reference and spacing_grids).
- ship's (or product's) basic geometry such as ship point, curve & surface, (see Building Blocks Moulded_form_point, Moulded_form_lines and Moulded_form_surfaces).
- ship's (or product's) configuration management (see Building Blocks versions, changes & approvals).

[8] More may be added in the future

[9] To be completed

- item (ship) itself are required since all data defining the product need to be related to the ship, which might exist in any life cycle stages (see Building Blocks ships);
- ship's units; the types of measures to be used such as metres, millimetres etc., (see Building Blocks measures).
- references to other models and instances (Building Blocks documents & external references).

Again the contents of each of these may or may not be used by every AP, but should an AP require such information then these constructs should be used rather the creation of new or disparate ones.

5.3 Building Blocks of the Ship Common Model

The following Building Blocks constitute those required to construct the full architecture of the Ship Common Model for all shipbuilding APs. Later sections describe what each of these Building Blocks contribute. The list currently includes:

- approvals
- changes
- connection_topologies
- date_time_resources
- definitions
- designation_characteristics
- dimension_characteristics
- documents
- events
- external_references
- generic_product_structures
- geometry_model_resources
- geometry_resources
- global_axis_characteristics
- interconnections
- local_co_ordinate_systems
- local_co_ordinate_systems_with_station_reference
- materials
- measures
- moulded_form_lines
- moulded_form_points
- moulded_form_surfaces
- organisation_resources
- p41_resources
- p42_resources

- p43_resources
- parts
- product_structure_by_assembly
- product_structure_by_system
- representation_resources
- ships
- spacing_grids
- support_resources
- topology_resources
- versions

Not all of these Building Blocks need to appear in each of the shipbuilding APs.

5.4 The Framework

Figure 7 describes the "backbone" of the different ARMs. It can be reused in the APs and specialised by sub-typing from the concepts presented and described in this section.

5.4.1 Items and Definitions

There are two major tasks when creating a product model:

- defining concept by specifying their properties
- describing how concepts are related to each other.

The properties of a concept are, in terms of modelling in EXPRESS, attributes of an entity. Also the relationship between two concepts could be modelled as an entity (such as a 'relationship-entity') with then two attributes of type 'entity'. In order to instantiate the 'relationship-entity' would therefore, require every that non-optional 'entity'-attribute be available.

This dependency is often not desired. It can be removed by separating the concept from it's properties - just from the modelling point of view, and allowing the concept to exist in an incomplete state until all of it's properties are specified (i.e. making it just a placeholder without attributes, but able to join relationships). Thus, in the Ship Common Model concepts may exist but the objects that they represent do not, until a definition of the properties have been defined. The SCM provides for this through the use of two constructs; **Item** (for concepts) and **Definition** (for properties).

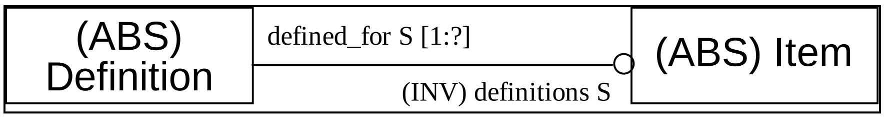


Figure 7: Relationship between Definition & Item

The concept (or placeholder) is called **Item**; the entity carrying the properties is called **Definition**. While a **Definition** must be defined *for* an **Item** (i.e. there is an existence constraint saying 'no **Definition** without associated **Item**') an **Item** may exist without any **Definition**s. This so called "defined_for" relationship shall be the only relationship between **Item**s and **Definition**s; no other attributes shall cross this boundary.

An **Item** has an attribute called "id". This is a life-time identifier of the thing. In some contexts this identifier is referred to as tag-number, this is an identifier for a function rather than for an instance.

The Ship Common Model currently distinguishes among the following **Definition**s:

- <Design_definition>
- <Functional_definition>
- <General_characteristics_definition>
- <Lightship_definition>
- <Loading_condition_definition>
- <Manufacturing_definition>
- <Parametric_definition>
- <Ship_material>
- <Structural_part_survey_definition>
- <Technical_description>
- <Tonnage_definition>.

This collection shows that the concept of **Definition** at least captures the following categories of properties of objects:

- characteristics
- functionality
- life-cycle.

5.4.2 Item Relationships and Item Structures

An **Item** can be instantiated without the need to instantiate any **Definition** - a data transfer that is based on this is complete from the instance model point of view (i.e. it is a valid data transfer), while being incomplete from the logical point of view (i.e. the thing which **Item** is the placeholder for, is not yet completely defined).

This allows us to create instances of **Item**s (the placeholders) and to describe their relationships (**Item_relationship**) on the same level of completeness.

Item_relationships may , for instance, be used to model that:

- one concept can be connected to another concept;
- one concept can be bounded by another concept;
- one concept can be derived from another concept.

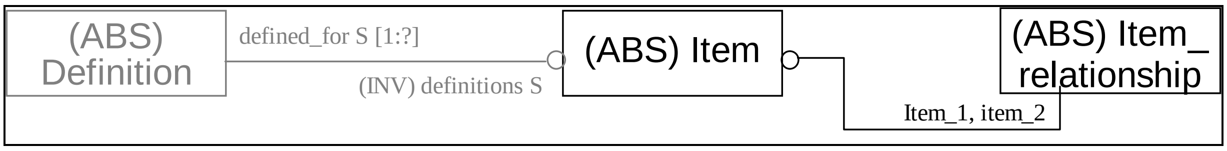


Figure 8: Item_relationship

Note that relationships that only carry existence constraints without any additional information need not be modelled by **Item_relationship**s, because existence constraints are implicitly provided by the EXPRESS modelling language. The following relationships may for instance better be modelled as attributes to **Item** than as **Item_relationship**s:

- one concept can be realised by another concept;

- one concept can be part of another concept;
- the existence of one concept can depend on the existence of another concept.

Besides **Item_relationship**s describing relationships between **Item**s there is a need for a container-like construct able to collect **Item**s and/or **Item_relationship**s from a specific point of view. Such a container is provided by **Item_structure** (see Figure 9: Item_structure) which functions in a similar way as **Item_relationship**. However, concepts (as well as their relationships) can be collected into an **Item_structure** without having any property defined, simply by collecting the placeholders of these things (i.e. the **Item**s and **Item_relationship**s) into such a container.

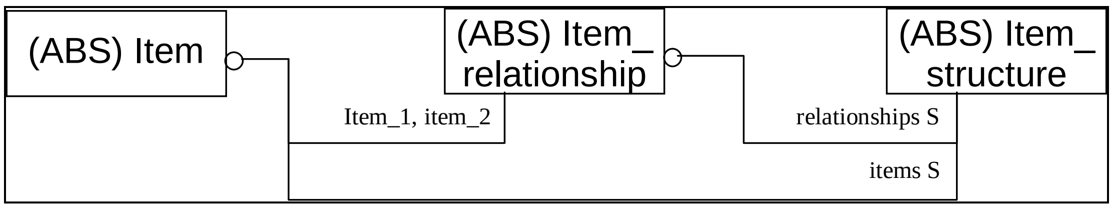


Figure 9: Item_structure

These three entities are located on the same level of completeness, i.e. no property has to be defined for the thing represented by its placeholder **Item** to say it exists, it may be related to another thing, and it is collected into a structure within a certain context.

In fact there is no reason for an **Item_relationship** not to have properties that are special to this relationship and that are not associated with the one or more concepts taking part in this relationship. This makes it desirable to allow **Item_relationship**s to have **Definition**s as well. And why shall an **Item_structure** not have properties special to this collection and not associated with the **Item**s or **Item_relationship**s it holds.

This requirement asks for a solution that allows **Definition**s to be defined for **Item**s, **Item_relationship**s or **Item_structure**s. The solution is to move this common behaviour one level up and let the three entities inherit the ability to be defined is this way . This is provided for in the SCM by the <Definable_object> construct as shown in Figure 10: Relationship between Items, Item_structures, Item_relationships & Definitions.

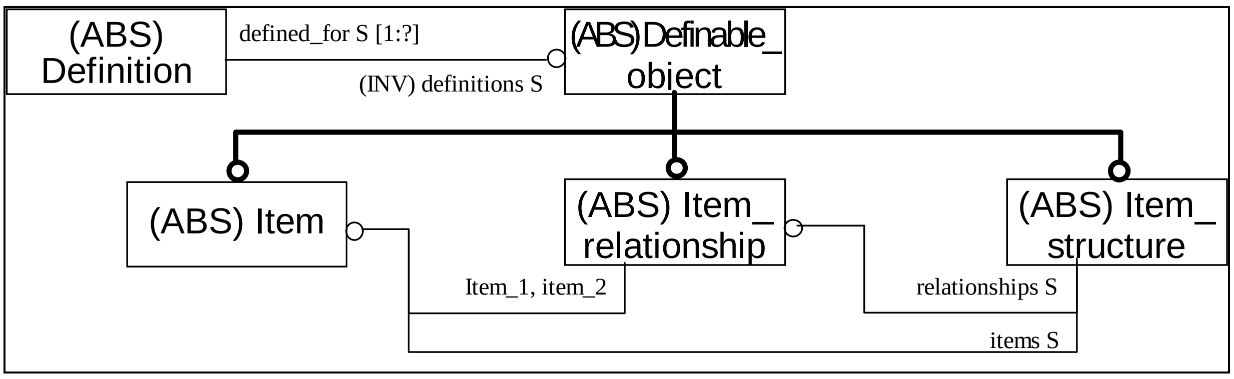


Figure 10: Relationship between Items, Item_structures, Item_relationships & Definitions

5.4.3 Representations

Every property of a concept, and therefore, every **Definition** of a **Definable_object**, may be described in many different ways. The definition of the shape of a plate for example, may be represented by its parameters length, breadth and thickness, or by a rectangular drawing and a thickness, or by a solid shape model. Each type of description may be useful in a different context. The drawing description could for example be useful for a production planning definition of the plate to be applied in nesting of pieces for flame-cutting. The solid description of the plate could relate to its design definition intended to support collision control.

In general, the detailed shape of a property shall be modelled as a **Representation**. However, if a few simple attributes are sufficient and if the usefulness of the description is not context dependent, properties may be referenced directly by a **Definition**. The parametric plate description is a typical example of this and is a candidate for direct inclusion into a **Plate_design_definition**. The description as a solid should be done by a **Representation**.

A typical relationship between a **Definition** and a **Representation** is shown in the figure below.

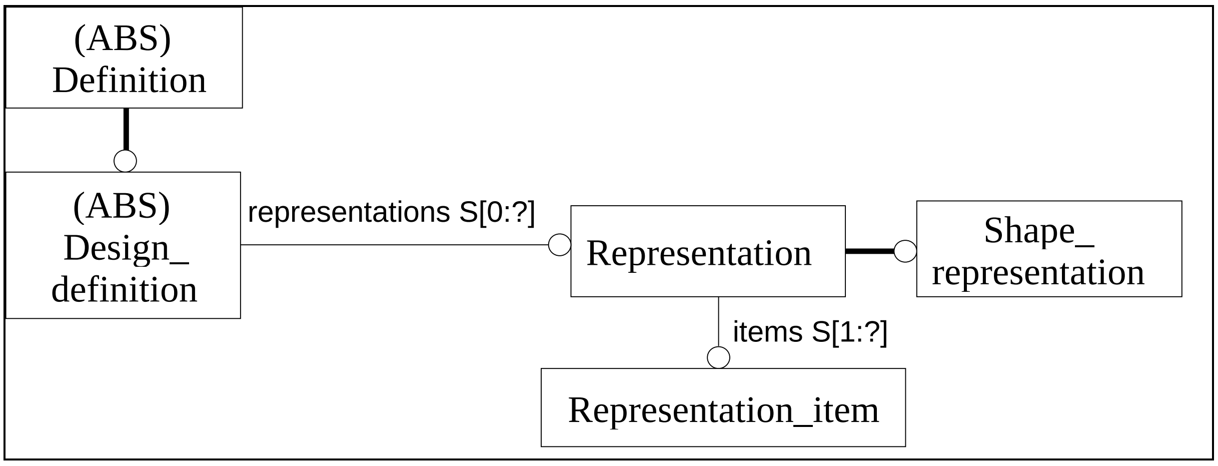


Figure 11: Relationship between Definitions & Representations

Other subtypes of **Definition** may be constrained to be certain subtypes of **Representation**.

The **Representation** concept itself is identical to the Representation in ISO 10303-43. Thus, the **Representation** of the Ship Common Model consists of a set **Representation_item**s. It is up to each **Representation** specialisation to constrain the valid types of **Representation_item**s. **Representation**s may also be related to each other with or without transformation by relationship entities.

The Ship Common Model currently distinguishes among the following **Representation**s:

- **Shape_representation**[10]

They are referenced by redeclararations (also see section 4.5.6) of the general relationship between **Definition** and **Representation** (see example 2).

5.4.4 Usage Guidance

For items and definitions;

- When defining sub types of <item> or <definition> care should be taken to restrict the use of complex entities. This should be done by the use of a WHERE rule to restrict the type to be "one of" the sub types rather than allowing the use of complex entities.
- The traditional mechanism for constraining the relationship between <item> and <definition> is through the redeclaration of the <defined_for> attribute. However, it is not always the case that a single <definition> will only be used for one <item>.

Example 1; a <parametric_feature_design> may be <defined_for> several lifecycle stages of a <feature>, therefore, it may be <defined_for> both a <manufacturing_feature> and a <design_feature>. It is also worth noting that many instances of an <item> may reference a single instanced of a <definition>. Hence, a number of instances of a <manufacturing_feature> (perhaps controlled by versioning) may all refer to a single instance of <parametric_feature_design>.

The following example shows how a design_definition is specialised for compartments and how the type of valid representations for the new entity can be constrained.

Example 2:

Compartment is an **Item**. One of the **Definition**s it may have is the **Compartment_design_definition**. The first attribute redeclaration below establishes this relationship. As **Compartment_design_definition** is a **Design_definition** and inherits the attribute representations, this can be redeclared in the second attribute statement below to only allow **Compartment_shape_representation**s to be valid **Representation**s of a **Compartment_design_definition.**

```
ENTITY Compartment_design_definition
  SUBTYPE OF (Design_definition);
(* ... *)
  SELF\definition.defined_for: SET [1:?] OF Compartment;
(* ... *)
  SELF\design_definition.representations: SET [1:?] OF Compartment_shape_representation;
(* the Compartment_shape_representation for which the Compartment_design_definition applies. *)
        ... ;
END_ENTITY;
```

The various parts described above can be shown below in the EXPRESS-G diagram presenting the contents of the three Building Blocks discussed above.

[10] Moulded_form removed as now a sub-type of representation. Hydrostatic_table is not a generic representation likely to be specialised by other AP's

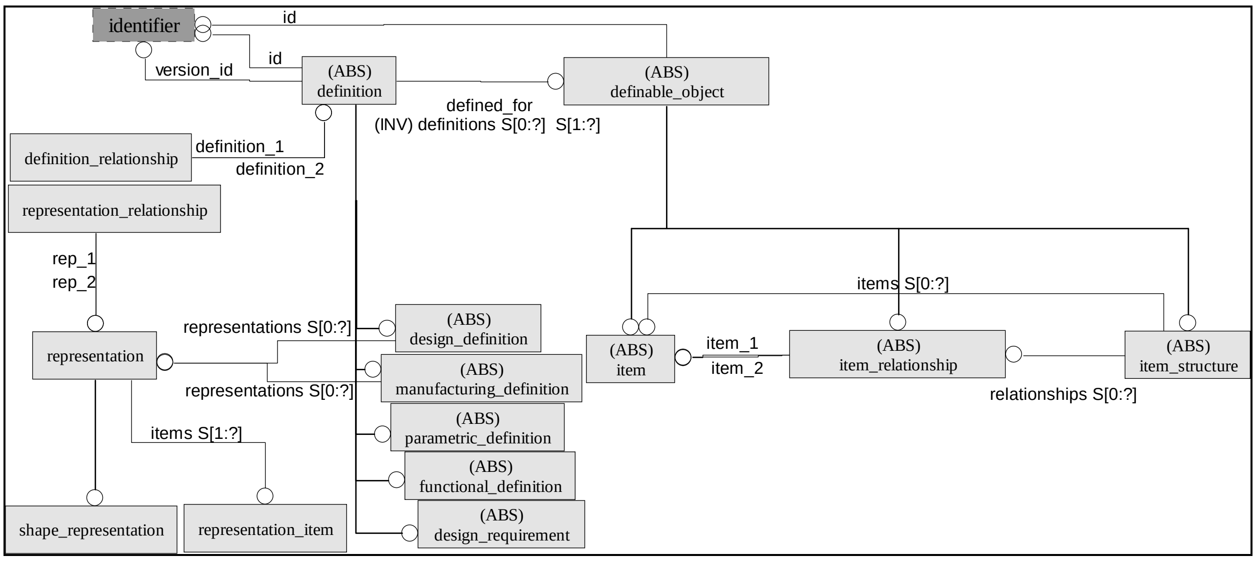


Figure 12: SCM Framework

5.5 Domain Models

5.5.1 Product Structure

It was found that in a number of places within the current shipbuilding APs that similar objects were related to each other and needed to be collected together under certain points of view. Furthermore there is a need to be able to create collections that consist not only of those objects, but also of other collections. The requirement is to create hierarchies of collections of objects and their relationships from different points of view. Examples are:

- The surface of the ship hull that may consist of a number of sub-surfaces (bow shape, bottom shape, parallel midship shape, stern shape, shape of appendices) related by topological and geometric continuity conditions, e.g. the starboard side parallel midship shape is a part of a mathematical plane bounded by (topological relationship) the bow shape, the bottom shape and the stern shape and has a G2 continuity condition (geometrical relationship) to them.
- The steel structure of a ship consists of blocks of groups of design panels of plates and profiles with the geometry derived from the ship interior and moulded form (geometric relationship), which are bounded by the interior and moulded form or by other structural parts (topological relationship) and which are welded together (joint relationship).
- The piping system of a ship consists of the fuel system, the fresh water system, the ventilation system ... where each of them may consist of subsystems of pipes, valves, flanges, pumps ... which are positioned with respect to the steel structure or to the interior or moulded form (topological relationship) and which are welded, screwed or whatever together and fitted to the steel structure (joint relationship).

In general, each collection of similar things and their relationships, appearing or getting realised (ideally or materially) during the life cycle of the product ship can be modelled as a product structure.

Common to all these examples is that there are single objects that are grouped together by several levels of collection hierarchy and that often are related to each other arbitrarily. It seems to be straightforward to model this in a common and generic way and to provide it for reuse.

This is the entry point to the Product Structure Concept. It is based on the three basic types:

- an **Item**, that are the objects of concern (the things),
- an **Item_relationship**, that sets two (or more) <Items> into a common context (the relationship between the things),
- an **Item_structure**, that collects **Item**s and **Item_relationship**s under a special point of view (the container for things).

These three entities are abstract that means not instantiable. However, their functionality to relate items and to collect items and their relationships can be reused by more special entities, that is by their subtypes.

5.5.1.1 Parts

A **Part** is an **Item** that does not really differ from it's parent. The reason for it's existence is to be able to restrict special **Item_structure** subtypes (such as **System** or **Assembly**) & to only collect **Part**s and not every **Item**. Without **Part**s, it would not be possible to make this restriction and by this e.g. an **Assembly** would be allowed to collect also every other kind of **Item** (such as **Moulded_form**, **Space**, **Ship** ...). The same is valid for the **Item_relationship** subtypes.

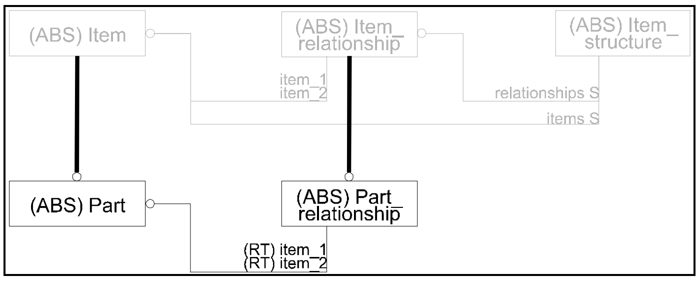


Figure 1.: EXPRESS-G of Generic Product Structure - Part Link

This means **Part** is the atomic **Item** in all **Item_structure**s collecting **Part**s.

Parts have been created by AP218, but can, of course, be reused by any other AP. If there is a need for another **Item** on this level, e.g. if other **Item_structure**s shall be restricted not to collect every **Item** and not either **Part**s, it must be created by the AP. An example is **Feature** of AP218.

There are actually four different product structures in use and more or less ready modelled by the shipbuilding APs. They will be described below.

5.5.2 Product Structure by System

Product Structure by System is intended to provide the abstract supertypes for different systems such as piping system, structural system, machinery system, electrical system ... and for their relationships.

A **System** is a function oriented, one-disciplinary view on a group of **Item**s in a way that only piping components or structural components ... are to be seen. It allows for a hierarchical, that is tree structure of **System**s. **System**s may consist of other (sub) **System**s. A **System** is both an **Item** and an **Item_structure**. WHERE rules in **System** ensure that a **System** can only consist of **Part**s and other (sub) **System**s.

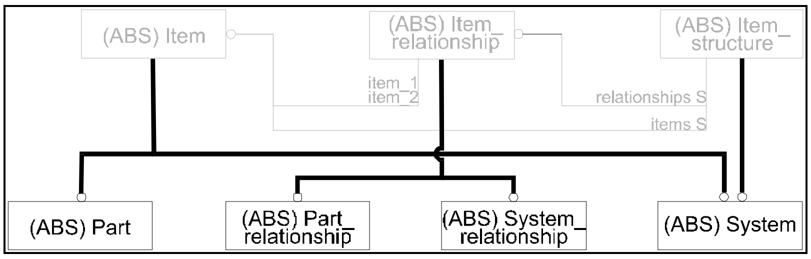


Figure 13: Product structure by system

5.5.3 *Product Structure by Assembly*

Product structure by Assembly specifies how the details of a ship, such as pipes, plates, machinery, cableways, shall be collected into greater units from the work preparation point of view. Such a unit is called an **Assembly**. Product Structure by Assembly gives the constructs for defining Assemblies.

Usually assemblies are cross discipline groupings consisting of pipes, structural parts (plates and profiles), machinery, cableways etc. Therefore, **Assembly** is not abstract, but ready for use.

It allows for a hierarchical, that is tree structure of **Assembly**s. **Assembly**s may consist of other (sub) **Assembly**s. An **Assembly** is both an **Item** and an **Item_structure**. WHERE rules in **Assembly** ensure that an **Assembly** can only consist of **Part**s and other (sub) **Assemblie**s.

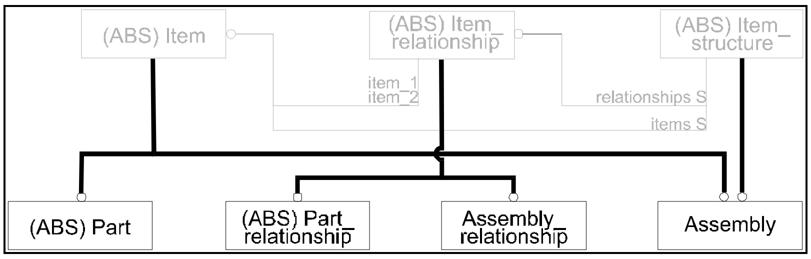


Figure 14: Product Structure by Assembly

5.5.4 *Product Structure by Space(Needs Input)*

<Does not exist - Input by Pete?>

Product structure by Space specifies how the spaces of a ship are related to each other.

Comment:

In opposition to the old implementation I would suggest to make <Compartment> to be the „<Part>“ of Product Structure by Space. This means <Compartment> is no longer a subtype of <Space> but now it should be an ***Item****. In this way it would be the atomic entity in a <Space>. <Space> should be restricted to collect <Compartments> and (sub) <Space>s. Having this a <Space> may consist of sub <Spaces> and/or of <Compartment>s, allowing a hierarchical tree of <Space>s with <Compartment>s at the tree path leaves.*

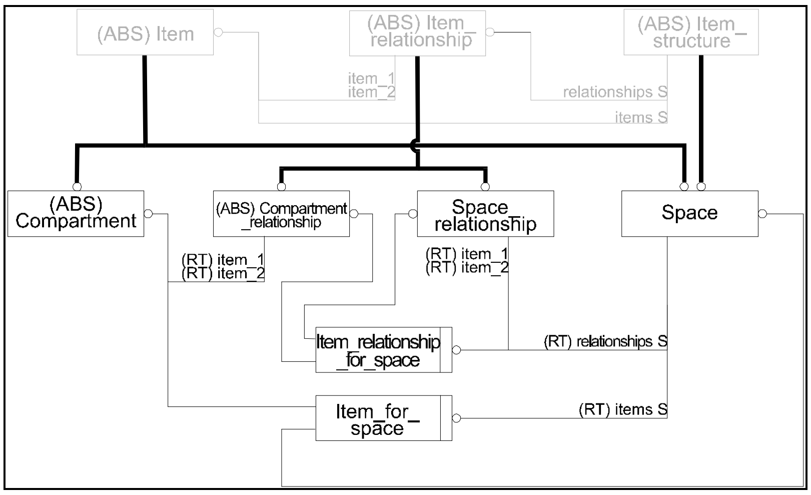


Figure 15:Product Structure by Space

5.5.5 Usage Guidance

5.5.5.1 How to **reuse** an existing Product Structure for an AP

1. Step: Decide which existing Product Structure meets your requirements.

2. Step: If needed create your own **Part**[11] subtype such as **Piping_part**. It may be non-abstract and, by this, instantiable, or abstract and, by this, supertype for more special **Part**s such as **Pipe**, **Valve** or **Flange**.

3. Step: If needed create your own **Part_relationship** subtype to be able to relate your newly created **Part**s. Restrict it only to be able to relate those **Part**s that meet your requirements by redefining the item_1 and item_2 attributes inherited from **Item** via **Part**.

4. Step: If the structure entity of the chosen Product Structure is abstract (such as **System**), create your own non-abstract subtype (e.g. **Piping_system**). Restrict it only to be able to collect those **Part**s and **Part_relationship**s that meet your requirements by adding WHERE rules to the newly defined Product Structure subtype applying to the items and relationships those attributes inherited from **Item_structure** via e.g. **System**.

5. Step: Create all desired **Definition**s for the newly created **Part**s such as **Pipe_functional_definition**, **Pipe_design_definition**, **Pipe_manufacturing_definition** ... Restrict them only to be plugged to those **Part**s that meet your requirements.

5.5.5.2 How to create a new Product Structure for an AP

1. Step: If needed make your own direct **Item** subtype such as **Compartment**. It may be non-abstract and, by this, instantiable, or abstract and, by this, supertype for more special Items such as **Tank**, **Cargo_compartment** or **Habitable_compartment**.

2. Step: If needed, make your own **Item_relationship** subtype (such as **Compartment_relationship**) to be able to relate your newly created **Item**s. This may again be either abstract

[11] „Part“ is only the placeholder. It may be substituted by every Item of the same level. At the moment only Part exists on this level within the Ship Common Model. Others could be Compartment or Feature (the latter still exists but is not SCM member)

or non-abstract depending on the usage intent. Restrict it to only be able to relate those **Item**s that meet your requirements by redefining the item_1 and item_2 attributes inherited from **Item_relationship**.

3. Step: Create your own **Item_structure** subtype (such as **Space**). If you wish to be able to have an implicit tree structure, make it also subtype of **Item**. Restrict it only to be able to collect those **Item**s and **Item_relationship**s that meet your requirements by adding WHERE rules to the newly created **Item_structure** applying to the items and relationships attribute inherited from Item_structure.

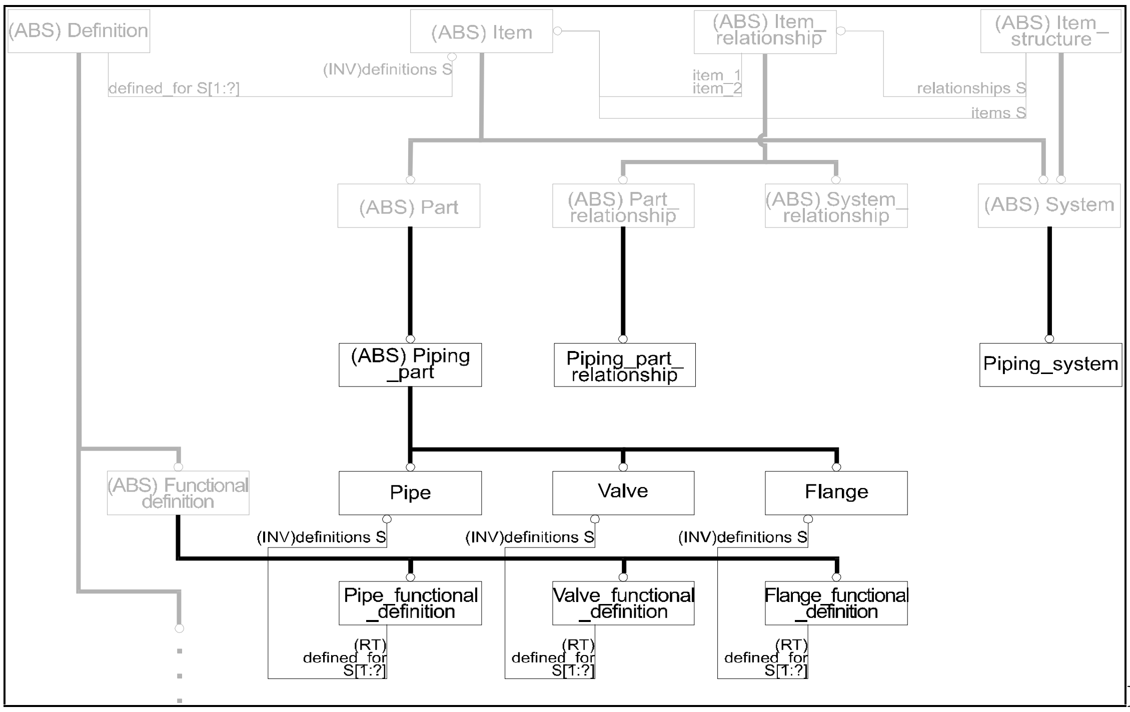


Figure 16: Reuse of an existing Product Structure for an AP

4. Step: If needed and if your newly created structure is subtype of both Item and **Item_structure**, make your own **Item_relationship** (such as **Space_relationship**) for relating your structure to other **Item**s or **Item_structure**s. Restrict it only to be able to relate those **Item**s that meet your requirements by adding WHERE rules to the newly created **Item_relationship** applying to the item_1 and item_2 attributes inherited from **Item**. Make sure, using another WHERE rule, that at least one of the related **Item**s is of your newly created structure type. Also make sure, using a WHERE rule, that the newly created structure type cannot be a substructure (neither direct nor indirect) of itself (acyclic see section 5.6.10).

5. Step: Create all desired **Definition**s for the newly created **Item**s such as **Compartment_functional_definition**, **Compartment_design_definition**, **Compartment_manufacturing_definition**, **Space_functional_definition**, **Space_design_definition**, **Space_manufacturing_definition** ... Restrict them only to be plugged to those **Part**s that meet your requirements by redeclaring the defined _for attribute inherited from **Definition**.

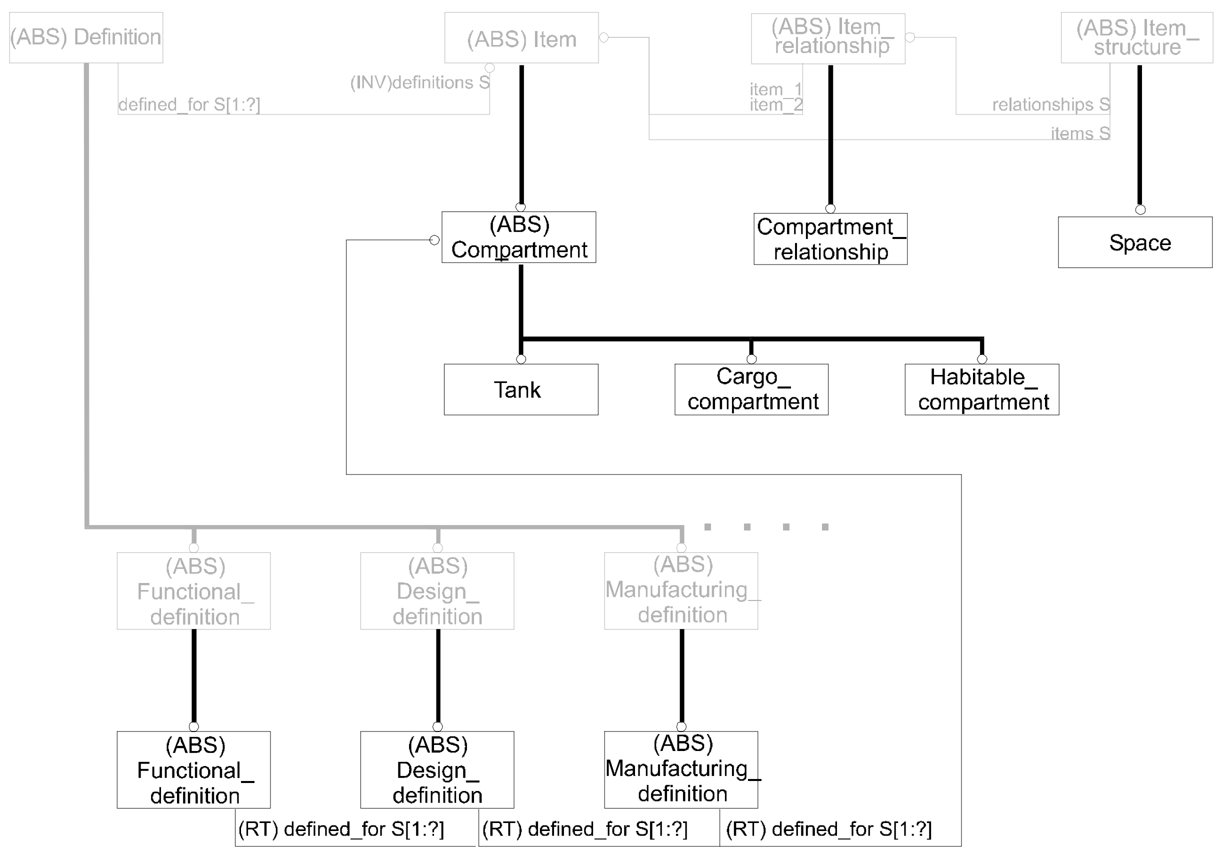


Figure 17: Creating a new Product Structure for an AP

The overall basic structure of the SCM, has now expanded the item side of the architecture as shown below in Figure 19.

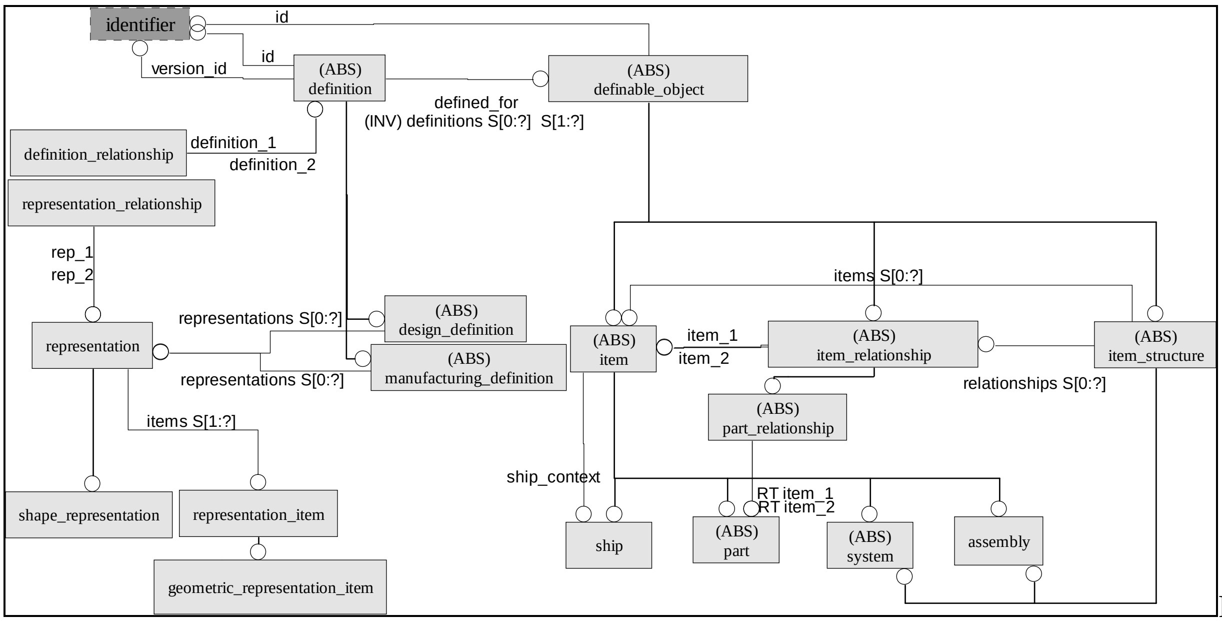


Figure 18: SCM Framework + Domin Models

5.5.6 *Connectivity(Needs Review)*

The Connectivity domain model is a reusable structure able to be specialised to suit the physical connections of the shipbuilding AP's. The basic structure is that of a relation (**connection**) which relates a number of **port**s together in a similar way that an arc connects two nodes (see section 5.6.10).

5.5.6.1 Connectivity Basic Model

Connectivity is a primary characteristic of any distribution system. In this section the fundamental connectivity structure is that of an "interface" connecting two or more "ends" or "ports". These ports are defined as potential connecting locations for a type of Item, as depicted in Figure 19. Using this basic model, the reader will observe that a network or graph can be easily represented.

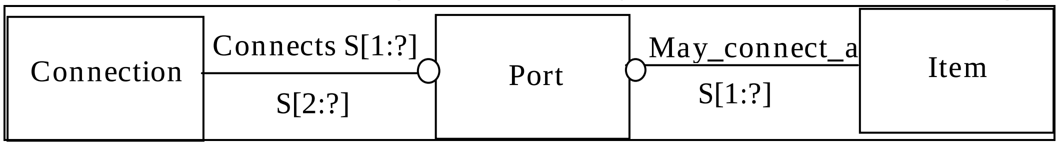


Figure 19: Basic Connectivity Model

From the basic connection concept, two specialised interpretations can be produced. These are the "interface" and the "interconnection", defined respectively in sections 5.5.6.3 and 5.5.6.4, below.

5.5.6.2 Ports

A Port is a possible connection location for the item with which it is associated. Piping ports are further classified by type (e.g. "flanged") such that connection type compatibility can be checked, along with checks for size and orientation compatibility.

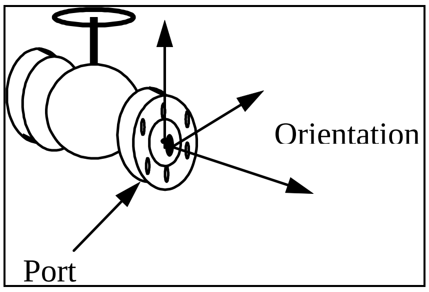


Figure 20: Ports

5.5.6.3 Distribution Interfaces

A Distribution_interface is a connection between two or more ports that are distribution_ends. It is a type of connection that joins two or more ports that are potential connection locations of *different* items. A Distribution_interface is a logical connection that has no shape or shape properties such as length.

EXAMPLE 1 - The joining of two flanged pipe ports, each belonging to a separate pipe flange, is an example of a Distribution_interface (that is, additionally, a pipe_joint). The Distribution_interface is the primary mechanism by which different items in a distribution system are connected into a distribution system.

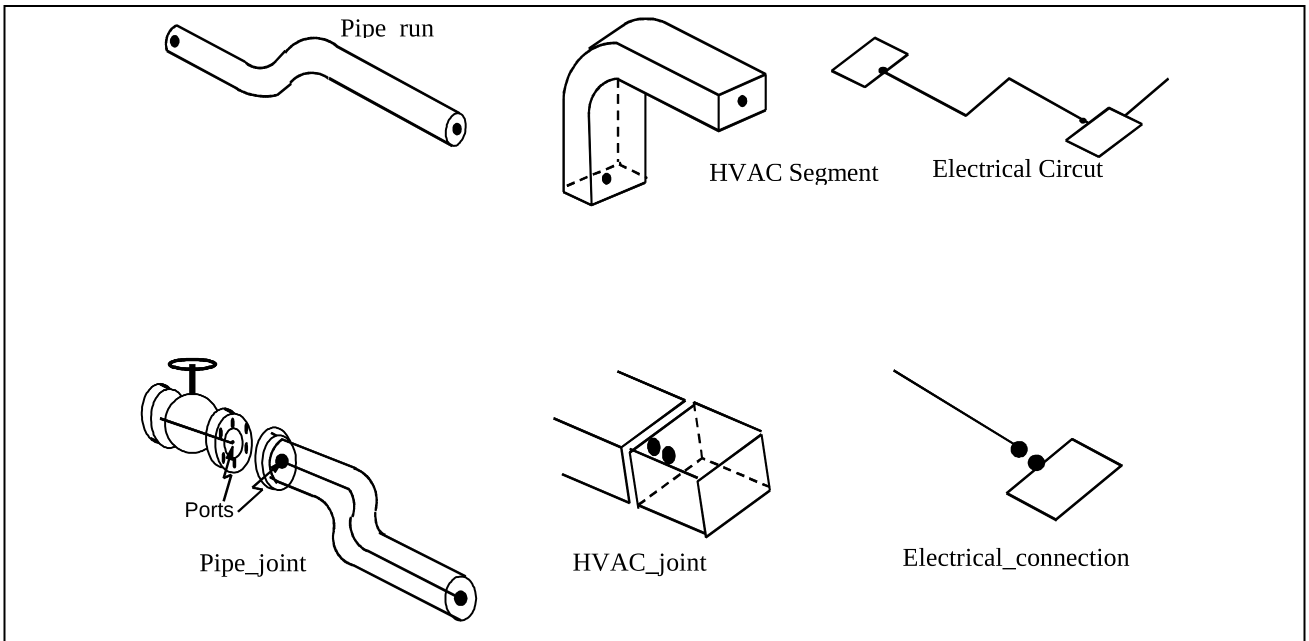


Figure 21: Distributions Interconnections & Interfaces

5.5.6.4 Distribution Interconnections

A Distribution_interconnection is an *item* whose role in a distribution system is to provide connectivity between two or more geographically separated other items. In contrast to a Distribution_interface, a Distribution_interconnection has shape and shape properties, and may have a sub-product structure, i.e. it may contain parts or other Distribution_interconnections. The concept of a Distribution_interconnection can then be specialised depending upon the domain. For example, within Ship Piping, Distribution_interconnection specialised in two ways; e.g. the Pipeline and Pipe_run. Figure 21 depicts examples of both Distribution_interfaces and Distribution_interconnnections in the different distribution system type domains.

5.6 Common Utilities

5.6.1 Ship General Characteristics

Concepts for general characteristics to be used within any Ship AP are provided by the following Building Blocks:

- designation_characteristics
- dimension_characteristics

General characteristics properties constitutes the basic information regarding details of the ship's dimension and identification. This information is independent of any geometric context. It includes scalar values and identification labels for principal dimensions of a ship, designation information for ship related companies, as well as class notation and references to relevant rules and regulations. If this information is not totally consistent with that which can be derived from the ship's geometric representation, then the latter, i.e. the geometrically derived information that shall have precedence.

While designation_characteristics BB contains the data definitions about the ship based from the owners, the yards and the classification societies' point of view, dimension_characteristics BB provides principal dimensions as well as class dimensions. Additional entities not related to geometry, e.g. keel parameters, overall dimensions and propeller parameters are specified in the moulded_form_characteristics BB, which is currently not part of the Ship Common Model.

The above mentioned BBs contain the following main elements (ordered alphabetically):

- Class_and_statutory_designation,
- Class_notation,
- Owner_designation,
- Regulations,
- Ship_designation,
- Shipyard_designation,
- Class_parameters, and
- Principal_characteristics.

A <Class_and_statutory_designation > is a type of <General_characteristic_definition> that specifies the identification given to the ship by the classification society for the purpose of design, manufacture and in service approval.

A <Class_notation> is the collection of classification rules that are being used to assess the design, manufacture and in service maintenance of the ship.

An <Owner_designation> is a type of <General_characteristic_definition> that specifies the organisations that own, or are involved with managing, the ship.

<Regulation>s provide a set of all international and national regulations as well as standards which apply to the ship.

Error! No topic specified.

Figure 22 - Designation characteristics BB

<Class_parameters> are a type of <General_characteristic_definition> that specifies the length and speed of the ship in accordance with Classification Society rules and statutory regulations.

<Principal_characteristics> are a type of <general_characteristics_definition> that describe the main shape parameters of the hull moulded form. <principal_characteristics> also includes data that is required in subsequent iterations of the hull development process when one is considering hydrostatics.

A <Ship_designation> is a type of <general_characteristics_definition> that specifies the identification given to the ship in order that it can be categorised by any shipping related organisation.

A <Shipyard_designation> is a type of <General_characteristic_definition> that is the identification given to the ship by the shipbuilder.

Error! No topic specified.

Figure 23 - Dimension characteristics BB

5.6.2 Configuration Management (To be completed)

5.6.3 Location concepts

Within any Ship AP location concepts are required to define the position of a whole ship or any of it's components unambiguously in global 3D space. To accomplish this, a number of entities are provided by the following Building Blocks:

- global_axis_characteristics (See Figure 24),
- local_coordinate_systems,
- local_coordinate_systems_with_station_reference, (See Figure 25 for both) and
- spacing_grids (see Figure 26: Spacing Grids BB.

The most important location concepts are co-ordinate systems and spacing grids. The major difference between them can be described as follows. Co-ordinate systems are mainly used to specify the exact location of moulded forms, structural parts, systems, pipes, compartments and spaces etc. directly in terms of geometry. They can be related to ship-specific positions (stations). Spacing grids use a ship-specific notation to position, (e.g. structural parts) along the major axes of the ship.

Within the context of Ship APs it seems to be sufficient to only support Cartesian co-ordinate systems (and of course spacing grids), i.e. systems with three orthogonal, normalised axes forming a right-handed system. Even if a special application would internally use a non-Cartesian system, e.g. with skew axes, this information probably does not to be exchanged as this is specific to that application.

The above mentioned BBs contain the following main elements (application objects):

- Global_axis_placement,
- Local_co_ordinate_system,
- Local_co_ordinate_system_with_station_reference,
- Station_reference,
- Location_reference,
- Spacing_position,
- Longitudinal_position,
- Spacing_table, and
- Spacing_grid_definition,

- which are explained in more detail below.

Any coordinate system is either a <Global_axis_placement> or a <local_coordinate_system>. Geometric data can be specified in terms of either one or the other.

A <global_axis_placement> is a type of <general_characteristics_definition> that defines a fixed system of right handed orthogonal axes to which geometric data are referred. A <global_axis_placement> shall have a:

- positive z axis in an upwards direction starting from the base of the ship,

- positive x axis running along the ship on the intersection of the centerline with the base and is in one case directed from the after part of the ship to the forward part of the ship or in the other case is directed from the forward part of the ship to the aft part of the ship,

- origin of the global axis placement can be any point on the x axis.

The distance of the after perpendicular from the origin and the orientation of the x axis shall be specified.

If any other system of axes is used, local or global, then the transformation relations between it and the <global_axis_placement> shall be specified.

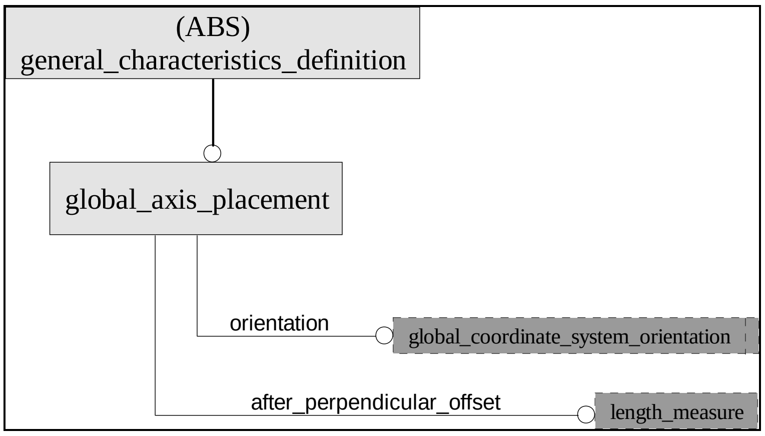

(ABS)
general_characteristics_definition
global_axis_placement
orientation
global_coordinate_system_orientation
after_perpendicular_offset
length_measure

Figure 24: Global_axis_characteristics BB

A <local_coordinate_system> is a type of <Definition>, which means, e.g. that a versioning mechanism is applicable. It is characterised by an origin and the location of it’s 3 axes. An arbitrary number of <local_coordinate_system>s may exist. Each <local_coordinate_system> has a parent system, which is either a <local_coordinate_system> again or a <Global_axis_placement>. Consequently, all coordinate systems within a model are structured in a hierarchy. The common root element of this hierarchy is a unique <Global_axis_placement>. Therefore geometry data defined in any <local_coordinate_system> can be transformed into co-ordinates of the <Global_axis_placement>.

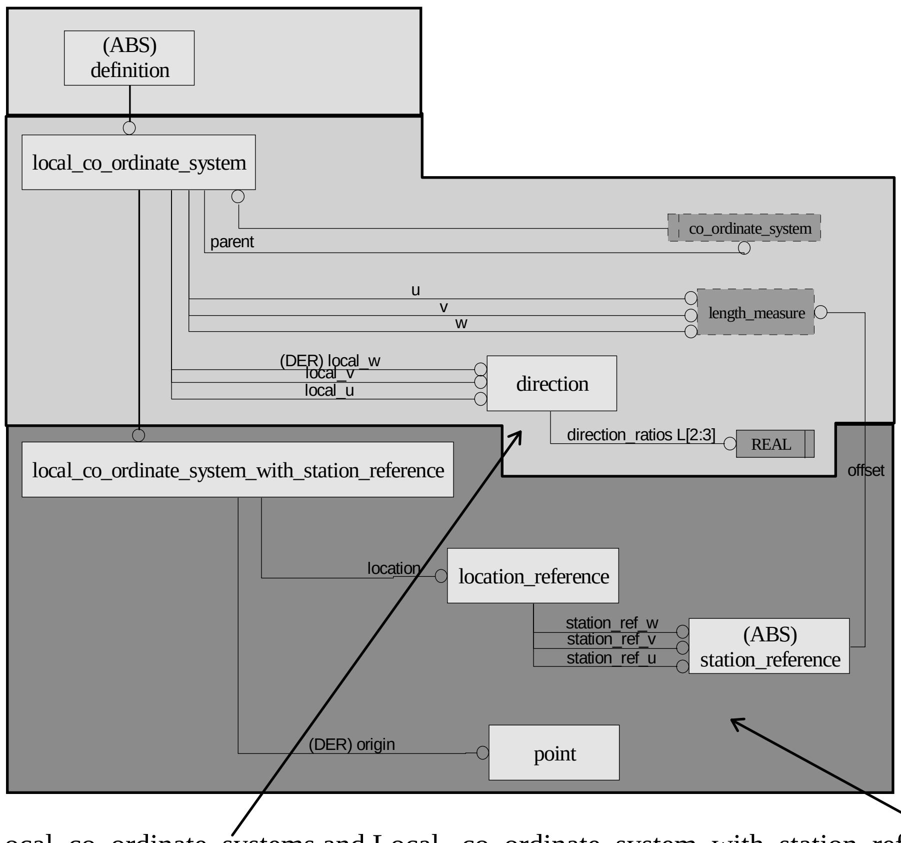


Figure 25: Local_co_ordinate_systems and Local _co_ordinate_system_with_station_reference BBs

A <Local_coordinate_system_with_station_reference> is a special <local_coordinate_system>, that allows for additional references to longitudinal, vertical or transverse stations. A <station_reference> is a particular entity, mainly used for this purpose. It also holds the information required to subdivide any axis into intervals which can be used as a reference basis for points in the axis system.

<Spacing_position>s may be defined for any of the three global co-ordinate system axes of the ship. It is used as a reference point during the design and manufacture of the ship and characterised by name, position and location.

Error! No topic specified.

Figure 26: Spacing Grids BB

A <Longitudinal_position> is a type of <Spacing_position> that specifies a location on the x-axis of the ship coordinate system. A <Longitudinal_position> may be defined by an offset distance from an given <Spacing_position>.

A <Spacing_table> is a collection of <Spacing_position>s that define a list of reference points along one of the coordinate axes of the ship. Any <Spacing_table> has a certain table_usage describing it's function within the context of the design and manufacture of a ship. Possible values for this usage are vertical_table, frame_table, waterline_table, etc.. Additional attributes are the name and a textual description what is the intended purpose of this <Spacing_table> .

Example: A frame spacing table is a special <Spacing_table>. The frame numbers would be specified as table_positions and the fact that the table was a frame spacing table would be given by the value for table_usage.

A <Spacing_table> may be created standalone or it may be referred to by a <Spacing_grid_definition>. The latter is a type of <General_characteristics_definition >. It describes the set of longitudinal, transverse and vertical <Spacing_table>s that are applicable to the ship. There is only one <Spacing_grid_definition> defined for a ship.

5.6.4 Basic Geometry

This section introduces three Building Blocks, moulded_form_points, moulded_form_lines and moulded_form_surfaces. These provide the basic ship geometrical concepts, <ship_point>, <ship_curve> & <ship_surface> respectively, which should be used in the shipbuilding AP's. It is intended that these BB's;

- moulded_form_points (<ship_point>),
- moulded_form_lines (<ship_curve>) &
- moulded_form_surfaces (<ship_surface>),

- should be used in preference to those of the generic resources[12].

All three Building Blocks have the same structure, but with different contents, see Figure 27. The intention of these Building Blocks is to extend the geometric resources from Part 42 with additional generic or shipbuilding specific information.

Example:

A b-spline curve, which can be any 2D or 3D curve, has no information about the context in which it will be used. If there is additional information related to this b-spline curve, for instance that this <curve> is a waterline, then this information implies that it is a 2D curve in x/y plane and used in a shipbuilding context. This information can be used by processors to derive additional information or for further constraints.

[12] Such as <point>, <line> & <surface>

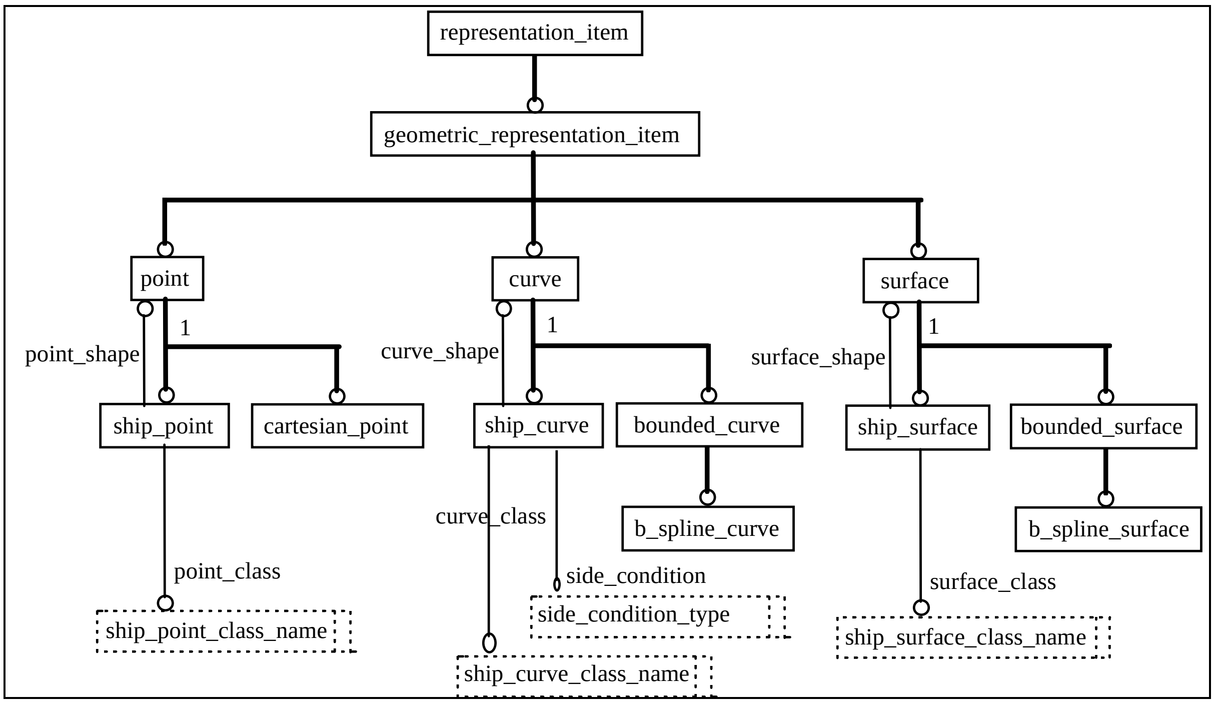


Figure 27: EXPRESS G of the Building Blocks moulded_form_points, moulded_form_lines and moulded_form_surfaces.

5.6.4.1 Moulded_form_points

The moulded_form_points Building Block introduces the Entity <ship_point>, which is a <point> in a specific context. The <ship_point> has two attributes, point_class and point_shape and is a Subtype of <point>, a Part 42 geometric resource.

<Point> is the supertype of all other geometric <point>s like <cartesian_point>, <point_on_curve>, <point_on_surface>. <Point> itself is a subtype of <geometric_representation_item> which is a subtype of <representation_item>. Because of the implicit declaration in the Cookbook, which is relevant for all shipbuilding AP's, (i.e. all subtypes are ONEOF by default), e.g there is no complex instance between a <ship_point> and any other geometric subtype of <point> like <cartesian_point> allowed to cover the geometric information for the <ship_point>.

Instead, the geometry of the <ship_point> will be covered by the attribute point_shape, which is related to <point>. A Where rule specifies, that no <ship_point> can be instantiated by itself through point_shape, i.e. no circularity is allowed.

The context of the <ship_point> is defined by the attribute point_class.

The following ship_point_class_names are supported:

– **ordinary** is a point of discontinuous tangency for one or more curves that pass through it.

– **tangent** is a point such that curves passing through it have a specified tangent.

– **knuckle** is a point that has no tangency information.

– **unspecified** is a point such that no tangency information is known or recorded.

Example: a <ship_point> which is a knuckle point can have the following attributes:

– ship_point_class_name *knuckle*

– point_shape is <*cartesian_point*> or any other geometric subtype of <point>

5.6.4.2 Moulded_form_lines

The moulded_form_lines Building Block introduces the Entity <ship_curve>, which is a curve in a specific context. The <ship_curve> has two attributes, curve_class and curve_shape and is a Subtype of curve, a Part 42 geometric resource.

Curve is the supertype of all other geometric curves like polyline, circle and b_spline_curve. Curve itself is a subtype of geometric_representation_item which is a subtype of representation_item. All subtypes of curve are ONEOF which means there cannot be a complex instance between a <ship_curve> and any other geometric subtype of curve like b_spline_curve to cover the geometric information of the <ship_curve>.

Instead the geometry of the <ship_curve> will be covered by the attribute curve_shape, which is related to curve. A Where rule specifies, that no <ship_curve> can be instantiated by itself through curve_shape.

The context of the <ship_curve> is defined by the attribute curve_class.

The following ship_curve_class_names are supported:

– **buttock_line** is a curve that is the intersection of a longitudinal plane with a hull moulded form.

– **centreline_profile** is a curve that is the intersection of the centreplane with the hull moulded form.

– **deck_line** is a curve lying on the moulded surface of a deck.

– **flat_of_bottom** is the boundary curve of the planar surface at the base of a ship.

– **flat_of_side** is the boundary curve of the planar surface at the outer most port or starboard side of a ship.

– **intersection_line** is a curve that is the intersection of two surfaces found on or within a moulded form.

– **knuckle_line** is a continuous boundary between two moulded form surfaces that has discontinuity in tangency across it. A knuckle line may pass through a number of knuckle points.

– **tangent_line** is a continuous boundary between two moulded form surfaces that has a specified tangent across it. A tangent line may pass through a number of tangent points.

– **transverse_section** is a curve that is the intersection of a plane that is parallel to the transverse axis with a moulded form.

– **waterline** is a curve that is the intersection of the water plane with a ship moulded form.

– **unspecified** is a curve whose relation to naval architecture is not known or not recorded

Example: a <ship_curve> which is a waterline can have the following attributes:

- ship_curve_class_name *waterline*
- curve_shape is *b_spline_curve* or any other geometric subtype of curve

5.6.4.3 Moulded_form_surfaces

The moulded_form_surfaces Building Block introduces the Entity <ship_surface>, which is a surface in a specific context. The <ship_surface> has two attributes, surface_class and surface_shape and is a Subtype of surface, a Part 42 geometric resource.

Surface is the supertype of all other geometric surfaces like plane, cylindrical_surface and b_spline_surface. Surface itself is a subtype of geometric_representation_item which is a subtype of representation_item. All subtypes of surface are ONEOF which means there cannot be a complex instance between a <ship_surface> and any other geometric subtype of surface like b_spline_surface to cover the geometric information of the <ship_surface>.

Instead the geometry of the <ship_surface> will be covered by the attribute surface_shape, which is related to surface. A Where rule specifies, that no <ship_surface> can be instantiated by itself through surface_shape.

The context of the <ship_surface> is defined by the attribute surface_class.

The following ship_surface_class_names are supported:

- **external_surface** is a surface that is, or forms part of, the hull moulded form.
- **internal_surface** is a surface that is, or forms part of, a structural element other than the hull moulded form, hull inlet or hull appendage

Example: a <ship_surface> which is an external_surface can have the following attributes:

- ship_surface_class_name *external_surface*
- surface_shape is *b_spline_surface* or any other geometric subtype of surface

5.6.4.4 Usage Guidance

This section concentrates upon how to use Building Blocks moulded_form_points, moulded_form_lines and moulded_form_surfaces in other AP’s.

It is the intention of the three Building Blocks moulded_form_points, moulded_form_lines and moulded_form_surfaces that they will be used in the shipbuilding AP’s in preference to the generic resources.

The Building Blocks were originally defined for the usage in AP 216 ship moulded forms. That means that the Enumeration lists of ship_point_class_name, ship_curve_class_name and ship_surface_class_name meet the requirements of AP 216. It is very likely, however that there are other types of <ship_point>s, <ship_curve>s or <ship_surface>s necessary for the different shipbuilding AP’s. The Enumeration lists have to be checked by every shipbuilding AP. If there is a new type of <ship_point>, <ship_curve> or <ship_surface> necessary, then the Enumeration list will be extended with this new type. To include this new type, a textual definition together with the name of the type has to be delivered. Each AP has to describe at least its own Subset of items in the enumeration lists, which is allowed for the data exchange.

<ship_point>s, <ship_curve>s or <ship_surface>s can be used by other AP's in any representation by redefining the attribute items of these entities. More useful is the definition of Where rules, which restrict the attribute items that it points to <ship_point>s, <ship_curve>s or <ship_surface>s. The attribute items will be inherited by any representation Subtype from the Supertype representation, which is part of the common model.

For instance in the offset table, wireframe or surface representations <ship_point>s, <ship_curve>s or <ship_surface>s can be used instead of the part 42 geometric resources.

Example: surface representation of the flat bottom using the Building Blocks moulded_form_points, moulded_form_lines and moulded_form_surfaces;

1. flat of bottom is a *moulded_form* which has a *moulded_form_representation*

2. the *moulded_form_representation* is a *face_based_surface_model*

3. the surface model consists of a *connected_face_set* with a single topological surface which is a *face*

4. the face_geometry of the *face_surface* is a *<ship_surface>*

5. the *<ship_surface>* has two attributes
   - surface_shape is a *plane*
   - surface_class is *external_surface*

6. the boundary of the *face* is an *edge_loop* consisting of one *oriented_edge* with one *edge_curve*

7. the geometry of the *edge_curve* is a *<ship_curve>*

8. the *<ship_curve>* has two attributes
   - curve_shape is a *polyline*
   - curve_class is *flat_of_bottom*

9. vertex1 and vertex2 are *vertex_point*s with the vertex_geometry *<ship_point>*

10. the *<ship_point>* has two attributes
   - point_shape is a *<cartesian_point>*
   - point_class is *ordinary*

5.6.5 Ships

- Ship itself is required since all data defining the product needs to be related to ship, which might exist in any life cycle stages (ships); it describes a ship as the naval architectural object in concern.

All data defining the product are somehow to be related to a ship, which might exist in any life cycle stage. A project, which represents a ship in the early design phase, for example before contract, is also regarded as a ship.

Error! No topic specified.

Figure 28 - Ships BB

5.6.6 Features(Needs input)

5.6.7 Materials (Needs Review)

This section introduces the Materials BB. The entities declared in this schema are used to specify a raw material by it's physical properties, not by the name it is sold under. This specifies the material's physical properties so that there is no dependence upon trade names for materials which may changes over time and from region to region.

The <ship_material>s can have a <description>, a <density_measure> and a <material_reference> defined externally. <ship_materials> are subdivided into <homogeneous_ship_material>s (with a number of physical properties) and <composite_ship_material>s which may consist of other <ship_material>s.

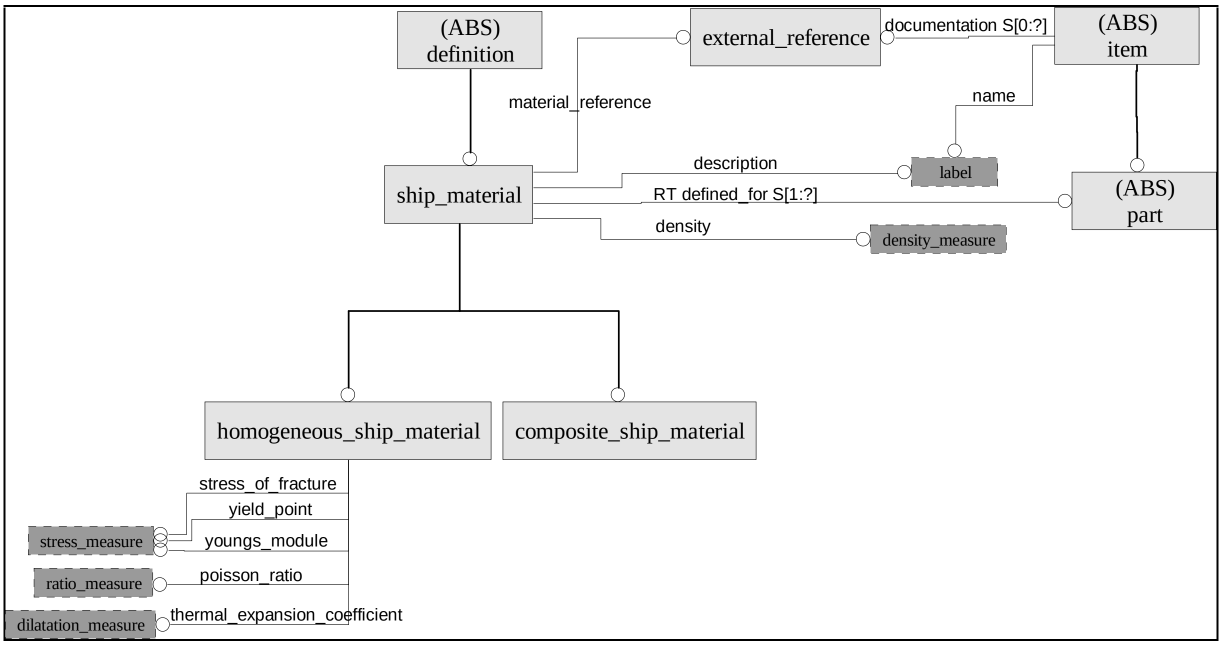


Figure 29: Materials BB

5.6.8 Units(Needs review)

- ship's units supported by the measures BB. <Measures> includes all resources that are required for representing measures for physical quantities. The following concepts are in scope:
  - global representation of units;
  - SI units;
  - derived units (such as for speed);
  - conversion based units (such as inch and foot);
  - all measures and units needed for all the ship APs.

The distinction between si_unit and conversion_based_unit is made by using the ANDOR relationship of the named_unit subtypes.

EXAMPLE:

A <length_measure> is ISO conformant if its unit is both a <length_unit> and a <si_unit>; it is a non-standard unit if it is both a <length_unit> and a <conversion_based_unit>.

END_EXAMPLE

This BB should be used by all shipbuilding APs. The concepts are taken from the IS-version of ISO 10303-41.

There are some minor changes compared to the Part 41 resources, such as

- to include new subtypes into the ONEOF clauses of existing supertypes;
- to include new types into existing select-types (measure_value);
- to include some additional WHERE-rules;
- to include type positive_number, percentage, force,
 pressure, stress, moment, inertia, dilatation ... .;
- to reference <unit> from the ship (=> no
 global_unit_assignment).

These lead to the creation of extended_... entities and to fewer imports from P41.

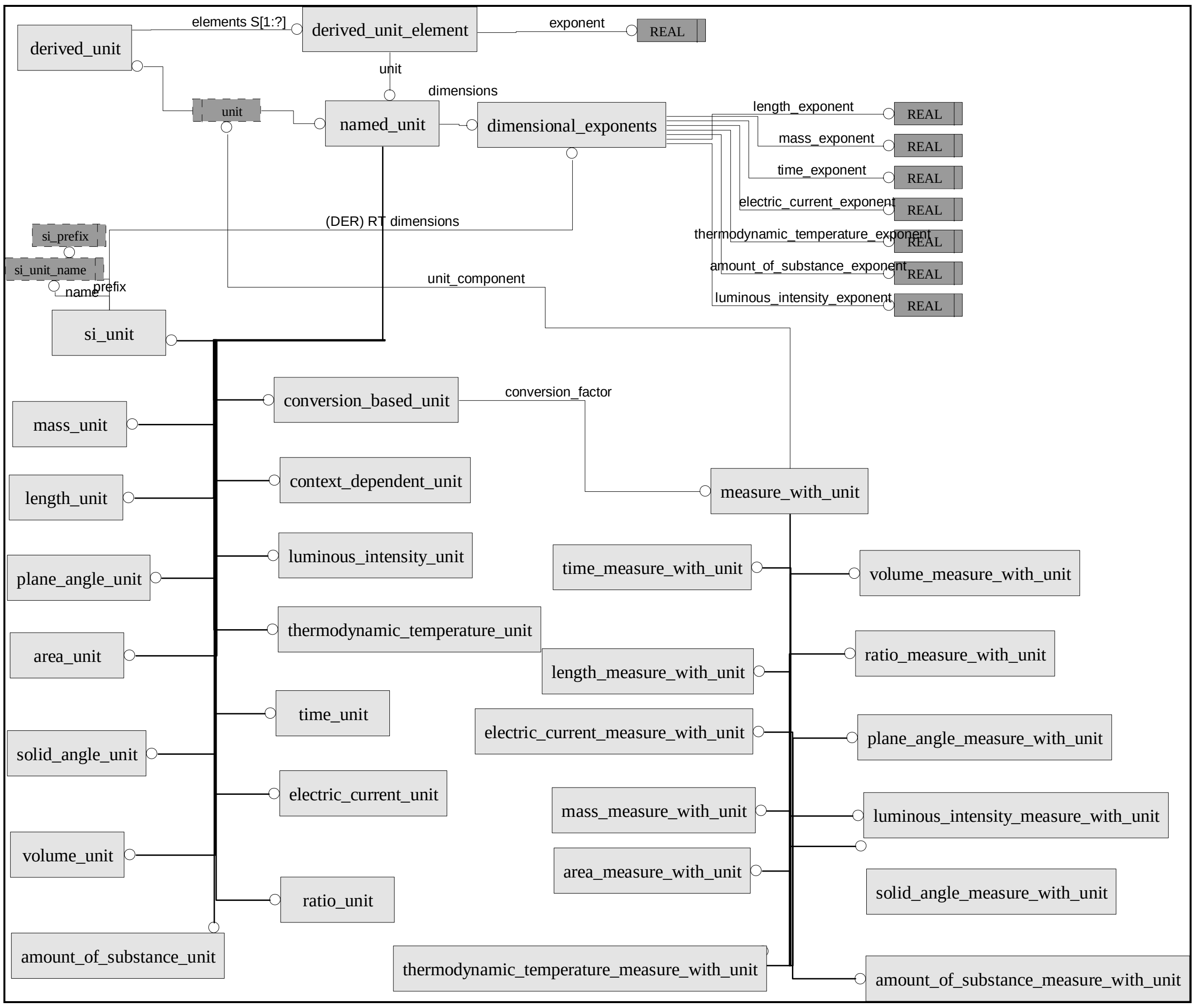


Figure 30: Measures BB

5.6.9 *Externally Defined References(Needs completing)*

This section is only partially complete. It introduces two utilities, that use similar terminology, but are otherwise quite distinct. Firstly, <External_reference>s are introduced, followed by <External_instance_reference>s.

5.6.9.1 External References

This section introduces the External_references Building Block which contains the entities to represent an <external_reference>. An <external_reference> is the abstract notion of a data source external to the data set where an instance of this entity exists.

EXAMPLE: a WWW uniform resource locator denotes such a data source

END_EXAMPLE

The information associated with an <external_reference> includes; a <location> and a <description>.

- the <location> provides a reference to the address where the source of the data can be found, such as an <address> whether postal or electronic.
- the <description> provides a textual note about the reference.

See Figure 5: Global_id & External References within the SCM below.

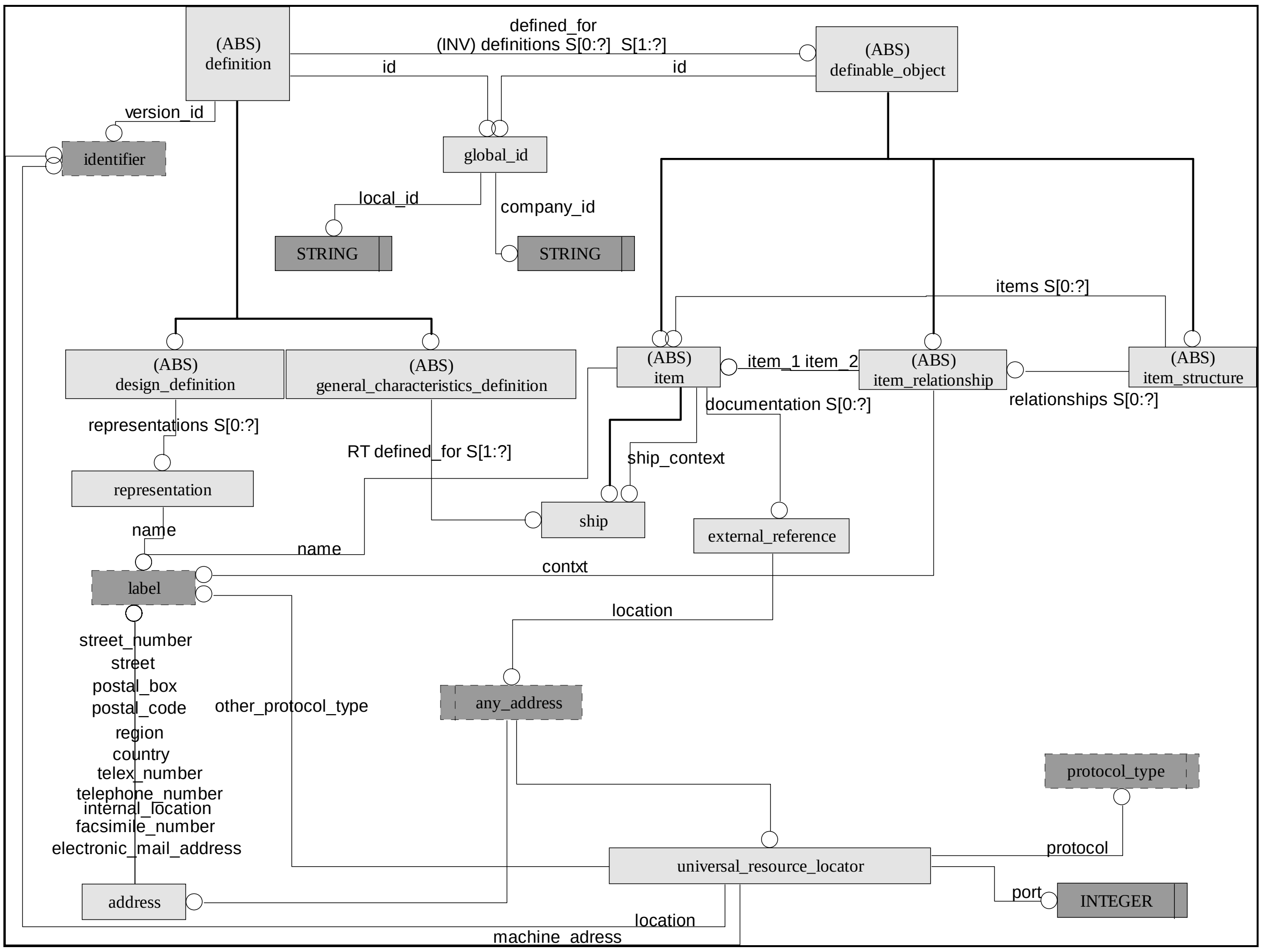


Figure 31: Global_id & External References in SCM

5.6.9.2 External Instance References

This section is incomplete but introduces the need for <external_instance_reference>s within the External_references Building Block and revisits the Definitions Building Block regarding the need for a <Global_id>.

<External_instance_reference>s provide the capability to refer to something outside a given data exchange or, in the data sharing context, to be able to reference instances of other models. In this

sense the notion of external instance referencing is similar to that of PLIB, and this issue is the concern of on-going work in the shipbuilding group.

External instance references are required for Shipbuilding APs for a number of reasons as explained within the modelling guidelines in section 4.9.1. For example, when needing to; preserve relationships between instances across different ISO APs; to preserve relationships between instances in the same ISO AP during partial transfers; to preserve the identity of instances which are transferred in multiple exchanges; to preserve the identity of entities during joint engineering tasks; or to preserve the identity of entities during concurrent engineering tasks.

Appendix A describes the current proposals for this facility, but as explained later, the exact format has still to be resolved. However, some issues[13] are still under discussion that will need to be reflected in this document.

Version information also is required in order to maintain consistency between transfers and environments. For example, a plate may reference a moulded form, but requires only the underlying surface geometry. If a reference to the moulded form is used then the referencing AP must use the moulded form schema to find the surface, i.e. the moulded form schema must be loaded and accessed. However, a direct reference to the surface would cause problems if the moulded form definition later referred to a different surface.

Only entity types which carry version information should be candidates for instance references. This includes all subtypes of “Definition”, and “Definable_object” (to capture item_structures and item_relationships).

Within the Definitions Building Block, a global unique identifier (Global_id) can be used to identify a unique version of a definition. Hence, a Global_id must be assigned when a new version of a definition is created. This permits concurrent updates to the same definition at several sites and can be seen in Figure 5: Global_id & External References within the SCM.

5.6.10 The Graphs Concept

This section has been included under the heading of utilities as many of the entities in the ship product model were based on graph or topological theory. This schema has since been dropped by the group, but the concepts behind the theory are still relevant and can be reused. The graph theory defines a set of relationships among objects.

Originally the graph constructs were collected in a Building Block of its own called graphs, but this approach, however, was considered too academic for use in the Application Reference Models and a decision was made to provide a description on how the concepts of the graphs could be used independently. The original graphs BB is listed at the end of this document, for reference purposes, within Appendix B.

The entities composing the former graphs BB would normally have been sub-typed for the new entities to inherit the attributes of the particular graph entity (& it’s parent entities). These former subtypes of the graph constructs now only show this historical fact in either comment notes or implicitly in the structure of the WHERE rules.

The basic concept is that of the <graph> entity which collects together a group of entities (such as <node>s), and a set of relationships <arc>s that describe which <node>s are related. It is in this sense that the topology or the arrangement of the <node>s can be described. The only constraints on

[13] (However, for a summary of the issues relating to this, see the minutes from the ISO Shipbuilding meeting held in San Diego, 1997, and also Annex G covering this topic).

this entity are that those <node>s in the set of <arc>s must also be in the set of <node>s. This means that there may be a <node> in the set of <node>s that is not referred to in the set of <arc>s. However, every <node> referred to in the set of <arc>s must be present in the set of <node>s. Of note, is that the order of the <node>s in an <arc> are not directed; i.e. the order of the <node>s representing node_1 & node_2 are not constrained.

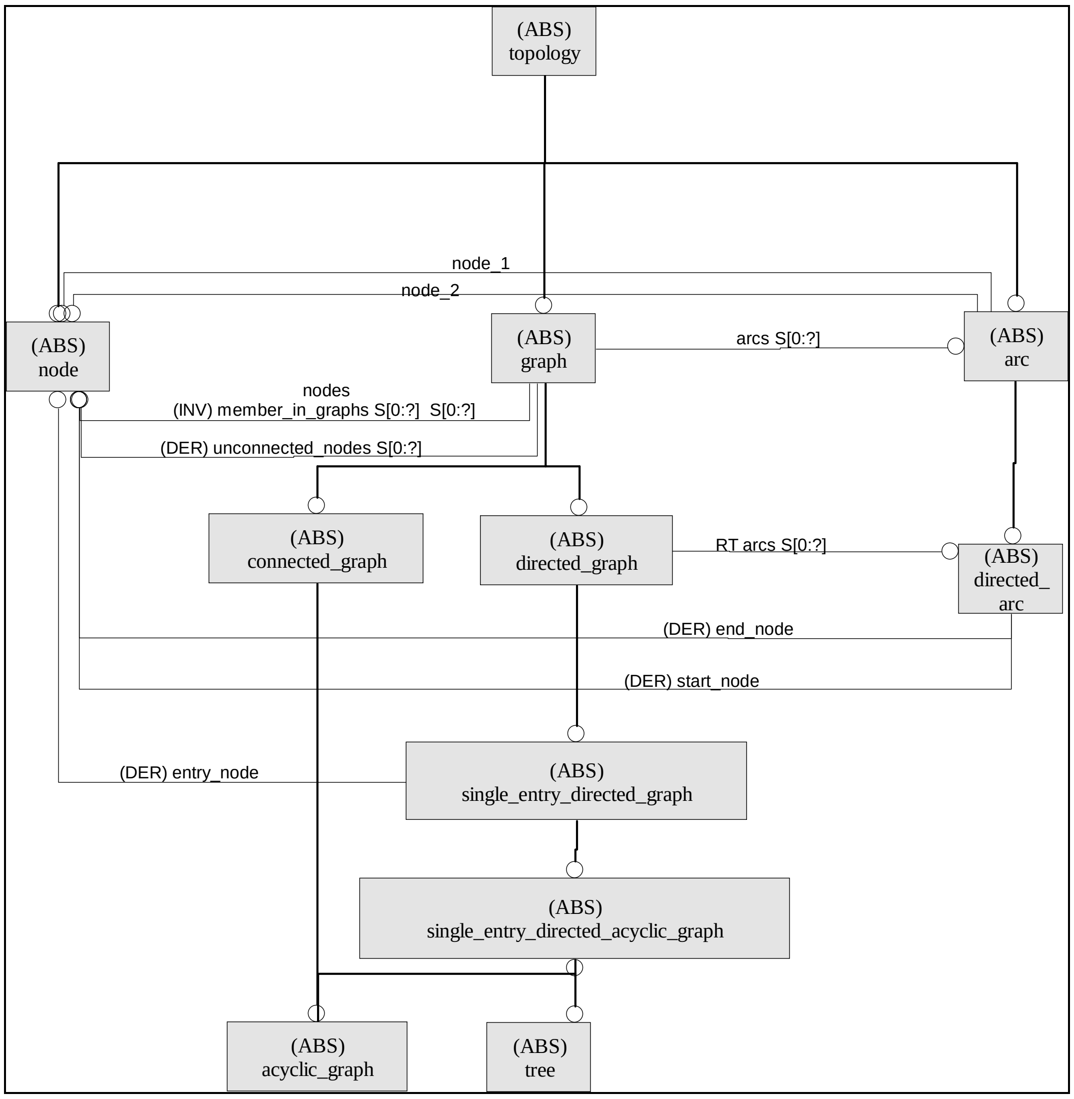


Figure 32: Graphs Building Block

```
ENTITY Topology
     ABSTRACT SUPERTYPE;
   -- <Topology> collects all major concepts of this schema.
  END_ENTITY;

ENTITY Graph
     ABSTRACT SUPERTYPE
     SUBTYPE OF (Topology);
    -- A <Graph> is defined as a related set of nodes and arcs. The
    -- following rules and constraints apply:
    -- 1. All nodes of the <Arc>s have to be nodes in <Node>s.
    -- 2. There may be nodes in <Node>s, which are not referred
    --    to by any arc in the set of <Arc>s.
    nodes: SET OF Node;
    arcs: SET OF Arc;
  DERIVE
    unconnected_nodes: SET OF Node := find_unconnected_node(nodes,arcs);
    -- the nodes which are not referred to in any arc in the set of
    -- <Arc>s.
  WHERE
    no_lost_arc_nodes: graph_condition_on_nodes(nodes, arcs);
```

```
      -- all nodes of the <Arc>s have to be nodes in <Node>s.
  END_ENTITY;


ENTITY Node
     ABSTRACT SUPERTYPE
     SUBTYPE OF (Topology);
  -- <Node> is a node in a graph.
  -- All application related entities which take part in
  -- relationships using the graph resource shall be a specialisation
  -- of this entity.
  INVERSE
     member_in_graphs : SET [0:?] OF graph FOR nodes;
  -- upward pointer for traversal of graphs.
  END_ENTITY;


  ENTITY Arc
      ABSTRACT SUPERTYPE
      SUBTYPE OF (Topology);
    -- An <Arc> represents a relationship between two <Node>s, without
    -- specifying the principal type of the relationship.
    --
    -- Application related entities which model a relationship between
    -- entities using the graph resource have to be declared SUBTYPE of
    -- this entity or any of its subtypes.
    node_1: Node;
      -- one node in the relationship, not necessarily the "first" or
      -- "primary" or "parent".
    node_2: Node;
      -- the other node in the relationship, not necessarily the
      -- "second" or "secondary" or "child".
  END_ENTITY;
```

By creating sub-types of graph, tighter and more restrictive relations can be defined. So for example, we might want to enforce the ordering of the information in the <arc>s to state that the parent should always be represented by the first node and the child by the second. This would be done by sub-typing graph (see directed_graph) and enforcing the <arc>s to be of type <directed_arc>, where the order of the <node>s is constrained.

```
ENTITY Directed_graph
     ABSTRACT SUPERTYPE
     SUBTYPE OF (Graph);
  -- A <Directed_graph> is a <Graph> where all arcs are directed.
  SELF\Graph.arcs : SET OF Directed_arc;
      -- all arcs of this graph shall be directed.
  END_ENTITY;

ENTITY Directed_arc
      ABSTRACT SUPERTYPE
      SUBTYPE OF (Arc);
    -- A <Directed_arc> represents a relationship between two <Node>s,
    -- in which case one node is viewed at as being the "first" or
    -- "primary" or "parent" node, the other node is by definition the
    -- "second" or "secondary" or "child" node.
  DERIVE
    start_node: Node := SELF\Arc.node_1;
      -- the "first" or "primary" or "parent" node.
    end_node: Node := SELF\Arc.node_2;
      -- the "second" or "secondary" or "child" node.
  END_ENTITY;
```

Likewise, we might want to enforce the explicit description of each child’s father and mother. In this case we must make it explicit that the father is a parent of the child (which was implicit in the previous example) followed by explicitly defining the mother. In this case all of the family members would be <node>s and the <arc>s would indicate that there was a relationship between both the father & the child and another relationship between the mother and the child. That is, two <arc>s would be needed to define a) the father-child relationship & a second b) the mother- child relation. In this sense the two <arc>s can then be considered to be “connected” by the fact that the child is acts as a common link between the two. The mother & father may also be the parent of another child, perhaps with a different partner, which would extend the length of the “connection” across a number of families. See the definition of a <connected_graph>.

```
ENTITY Connected_graph
    ABSTRACT SUPERTYPE
    SUBTYPE OF (Graph);
    -- A <Connected_graph> is a <Graph> where the following
    -- constraints apply:
    -- 1. all arcs have to be connected to form one single structure.
    -- 2. the set of unconnected nodes shall be empty.
    WHERE
      is_connected:   graph_is_connected(SELF\Graph.arcs);
      -- all arcs have to be connected to form one single structure.
      no_lost_nodes: SIZEOF(SELF\Graph.unconnected_nodes) = 0;
      -- the set of unconnected nodes shall be empty.
  END_ENTITY;
```

With a <connected_graph> described above, it would be possible to see that the “connection” described could continue to grow (as more & more partners are added to the list) until a partner eventually links with the original parent and effectively “closes the loop”.

This could be similar to a situation where (in ship building terms), we wanted to relate steel plates of a panel together. Each plate (or <node>) is related to another through a weld (perhaps a subtype of <arc>). The plates are connected because each weld (or arc) is common to two plates, and because each plate has a number of welds, it can be connected to a number of other plates. However, if we imagine that plates are being welded together around a sphere, sooner or later, the original plate will be welded to the plate leading the others around the sphere. In this sense, the <graph> of the plates would be cyclical.

[Add Figure to explain]

In certain situations this might not be desirable, hence the <acyclic_graph> below, restricts this through a WHERE rule.

```
ENTITY Acyclic_graph
    ABSTRACT SUPERTYPE
    SUBTYPE OF (Connected_graph);
    -- An <Acyclic_graph> is a <Connected_graph> with NO cycles.
    WHERE
      no_cycles: TRUE;
    -- graph_has_no_cycles(SELF\arcs);
    -- this graph shall be acyclic.
  END_ENTITY;
```

Returning to the description about directed_graphs, it can also be useful to constrain which of the <arc>s should be handled first. In a directed graph, each second <node> in one <arc> forms the second in the next <arc> to provide the “direction”. By specifying that the first <node> in the <arc> is not referred to by any other <arc>, then we constrain this <node> to have no other relationships in the <graph>. This might useful in situations where a single relationship provides the key to all others. For example, in genealogy, sometimes only one ancestor is known and acts as the root of the family tree. In the example below, the WHERE rule checks this condition.

```
ENTITY Single_entry_directed_graph
     ABSTRACT SUPERTYPE
     SUBTYPE OF (Directed_graph);
  -- A <Single_entry_directed_graph> is a <Directed_graph> which has
  -- exactly one start node referred to by only one arc.
  DERIVE
     entry_node: Node := find_entry_node(SELF)[0];
  WHERE
    entry_node_condition: (SIZEOF(find_entry_node(SELF)) = 1);
      -- this graph shall have exactly one start node referred to by
      -- only one arc.
  END_ENTITY;
```

This can also be extended towards the <acyclic_graph>, to ensure that there are no cycles elsewhere within the <graph>.

```
ENTITY Single_entry_directed_acyclic_graph
     ABSTRACT SUPERTYPE
     SUBTYPE OF (Single_entry_directed_graph, Acyclic_graph);
    -- A <Single_entry_directed_acyclic_graph> is a
    -- <Directed_acyclic_graph> which has exactly one start node
    -- referred to by only one arc.
  END_ENTITY;
```

Extending the idea of the family still further, the family tree can also be represented by a <single_entry_directed_acyclic_graph> with a corresponding WHERE rules to check that (in the family sense), a child can only have one father, and likewise, the child can have only one mother.

```
ENTITY Tree
   ABSTRACT SUPERTYPE
   SUBTYPE OF (Single_entry_directed_acyclic_graph);
  -- A <tree> is a <single_entry_directed_acyclic_graph> where no two
  -- arcs end at the same node.
WHERE
  is_tree: TRUE;
END_ENTITY;
```

5.6.10.1 Using the Graph Concept

As well as defining the constraints shown in the <graph>s BB, through the use of WHERE rules within shipbuilding entities, the supporting FUNCTIONs are also needed. The negative side to this approach is that there is a risk that similar functions are defined for use in the different BB’s, thereby trading greater clarity for a little redundancy in places.

However, we can see that by subtyping and constraining the basic <graph> definitions, more and more complex relationships can be represented. In the shipbuilding area, it is these rules that restrict the relationships between the various systems, entities of the framework and ultimately between the different ship parts, and can be used across many different domains within and outside shipbuilding. The graph schema and its concepts are used in the Ship Common Model already and serve to demonstrate the principles behind the use of topology.

6. QUALITY ASSURANCE

The following table states a minimum set of requirements to be matched by a Building Block ready for publication through the Building Block e-mail server. The criteria are ordered from the more vague and difficult to check to the more concrete and easy to check requirements.

Topic	Quality Target	Approach
Content of comments	understandability	a comment shall support together with the formal model constructs the understanding of the model concept. Construct and comment together shall facilitate the understanding and evaluation by a person not involved in the model development. See also section 4.4.6
Expressiveness of names	understandability	check if name chosen is meaningful with respect to the model intent
OPTIONAL attributes	consistency, clear model intent when used	see this document
EXPORT and IMPORT lists	leanness – avoid unnecessary imports and exports	check if an exported item is a potential candidate as a resource to be used elsewhere in the Building Block world. Check if each imported item is actually used in the model schema

Form of comments	compliance with commenting guidelines	see section 4.4
Form of entity and attribute names	compliance with naming restrictions	see section 4.4.5
Cardinality constraints	compliance with guidelines	see section 4.7
Use of ANDOR	no usage (neither implicitly nor explicitly) of the ANDOR or AND construct.	see section 4.5
EXPRESS-G	completeness	an EXPRESS-G diagram shall be present in the Building Block e-mail server
Form of Building Block	consistency	The Building Block shall conform to the form described in section 3.1.1
Syntax	correct EXPRESS IS syntax	parsing without errors

Table 1: Building Block requirements.

7. LIBRARY MAINTENANCE OF BUILDING BLOCKS AND ISSUES

There exists an automated storage and distribution mechanism for Building Blocks, issues and other types of data relevant for concurrent model development (the Building Block e-mail server). In order to ensure consistency and overall availability of these items, the server shall be used for their publication throughout the development process.

A current version of the usage guidelines for the server can be obtained by mailing *emsa@sdg.lr.org* stating *help* in the subject line.

The maintainer of the server is Tim Turner from LR in London, e-mail: tcstjt@aie.lreg.co.uk or tel: +44 181 423 2407

8. ISSUES

Current issues include:

1 Format of Global_id - is this sufficient for general use? (PLIB format should perhaps be used for later consistency).
2 Are any changes required to version history (maybe)
4 Should simple_version be retained (no)
5 Conformance class - how do we ensure conformance in an AP where external references are used to refer to definitions in other AP's ? (documentation)
6 Perhaps conformance classes are needed across AP's - therefore, we may need input from WG10? (Jochen)
7 Should the format of the GUID indicate whether referring to definition or definable_object?
8 If separate GUIDs are required, should we use the 2 attributes from the PLIB format?

9. BIBLIOGRAPHY

[N327] Koch Th., de Bruijn W.: ISO TC183/SC4/WG3 document N327 – Modelling Framework for Shipbuilding.

[AP Guidelines] Guidelines for the development and approval of STEP Application Protocols; version 1.2; ISO TC 184/SC4/WG4/ N506, May 5, 1995.

[LR12ADP] Turner, J.: STEP AEC Shipbuilding, AP Cross Reference List, October 17th 1997

[Common AAM] Kendall, J: ISO TC 184/SC4/WG3 N511 - Shipbuilding Common Activity Model, dated 14 January 1996.

[SCM-97] Turner, T: Ship Common Model, dated Version dated 30 July, 1997.

[Guidelines-96] Haenisch,J: ISO TC184/SC4/WG3/N498 - AP Development Guidelines for Shipbuilding, dated 17/07/96

10. APPENDIX A: CURRENT PROPOSALS FOR EXTERNAL INSTANCE REFERENCING

The EXPRESS-G diagram, together with the definitions below make up the current proposed during the San Diego (ISO) meeting June 1997.

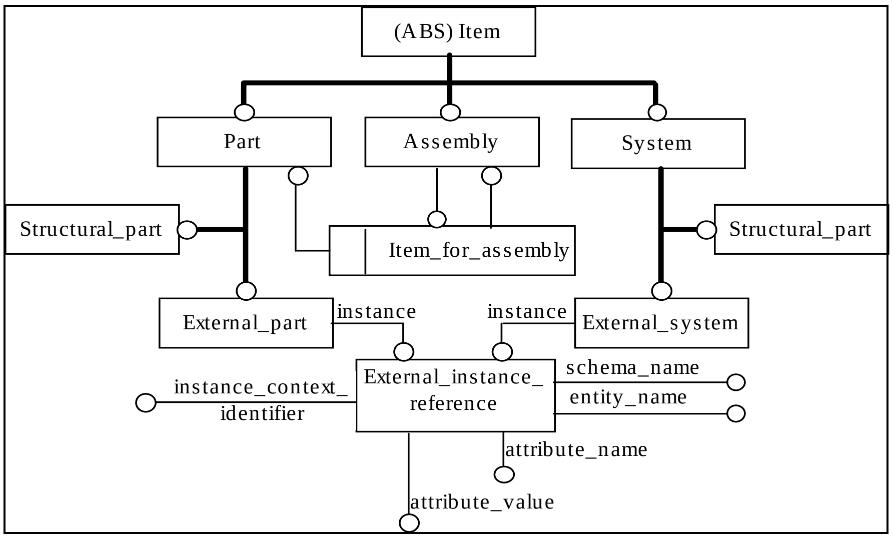


Figure 33: External References

The overall modifications required are now outlined below;

- the Generic_product_structures Building Block, the Definable_object entity needs to be re-modelled with external_reference properties/entity.
- the Definitions Building Block the entity *definition* should be modified as follows:

```
ENTITY Definition
    ABSTRACT SUPERTYPE;
    defined_for: SET [1:?] OF Item;
    local_units: SET OF Unit;
    version_id: OPTIONAL Label;
    GUID:   Global_id;  (*A persistent, global identifier which uniquely identifies the definition.  *)
    UNIQUE
    UR1:     version_id, GUID;
END_ENTITY;
```

The global identifier is modelled as an entity to allow access to the company_id field. The local_id is assumed to be unique within the company across all sites and projects.

```
ENTITY Global_id;
    (*  A persistent, global identifier which uniquely identifies the definition. *)
    company_id:   STRING (4) FIXED;
```

(* A unique identifier for the company which created the data. The string is left justified and blank filled. *)
local_id: STRING(64) FIXED;

(* A persistent identifier which uniquely identifies this definition throughout the company. Assigned at the time a definition is created. The string is left justified and blank filled. *)

END_ENTITY;

All external instance references will be subtypes of *External_Definition_Reference:*

```
ENTITY External_Definition_Reference;
   (* A reference to an definition which is external to this transfer *)
   schema_name:      Label;
   (* The name of the schema which contains the definition instance. *)
   entity_type:      Label;
   (* The type of the definition instance. This is a subtype of definition.*)
   GUID:             Global_id;
   (*A persistent, global identifier which uniquely identifies the definition.*)
END_ENTITY;
```

An Example using AP 215

Assuming that a previous transfer has already taken place & that the relevant GUID is available, then External instance references could be implemented in AP215 as follows:

```
TYPE moulded_form_boundary = SELECT
   (non_structural_moulded_form_design_definition,
   ap216_moulded_form_design_defintion);
END_TYPE;

ENTITY non_structural_moulded_form_design_definition
   SUBTYPE OF (Moulded_form_design_definition);
   (* a nonstructural moulded form.  These are specific to compartmentation. *)
   WHERE
   (* usage is restricted to non_structural or user_defined *)
   WR1: (SELF.usage = moulded_region_usage.non_structural_bulkhead) OR
   (SELF.usage = moulded_region_usage.user_defined);
END_ENTITY;

ENTITY ap216_moulded_form_design_defintion
   SUBTYPE OF (External_Definition_Reference);
   (* a structural moulded form which has been transferred using AP216 *)
   WHERE
   (* moulded form is from AP216 schema *)
   WR1:  SELF.schema_name = ‘ap216_maristep’;
   (* entity type is moulded_form_design_definition *)
   WR2:  SELF.entity_type = ‘moulded_form_design_definition’;
END_ENTITY;
```

11. APPENDIX B: THE COMPLETE GRAPHS BUILDING BLOCK

```
Graphs Building Block
Description:
  The schema graphs serves as a resource for all concepts based on
  dynamic (i.e. not fixed at modelling time) structures. It contains
  the basic element of a graph which is called "node". An "arc" is
  used to express a relationship between two nodes. A "graph" is
  basically a collection of arcs. Different types of graph are
  supplied to represent different types of relationships.

SCHEMA graphs_export;
  USE FROM graphs_model
          ( Node,
            Arc,
            Graph,
            Directed_arc,
            Single_entry_directed_acyclic_graph,
            Tree);
END_SCHEMA;

SCHEMA graphs_model;

  ENTITY Topology
     ABSTRACT SUPERTYPE;
   -- <Topology> collects all major concepts of this schema.
  END_ENTITY;


  ENTITY Node
     ABSTRACT SUPERTYPE
     SUBTYPE OF (Topology);
   -- <Node> is a node in a graph.
   -- All application related entities which take part in
   -- relationships using the graph resource shall be a specialisation
   -- of this entity.
  INVERSE
     member_in_graphs : SET [0:?] OF graph FOR nodes;
   -- upward pointer for traversal of graphs.
  END_ENTITY;


  ENTITY Arc
      ABSTRACT SUPERTYPE
      SUBTYPE OF (Topology);
    -- An <Arc> represents a relationship between two <Node>s, without
    -- specifying the principal type of the relationship.
    --
    -- Application related entities which model a relationship between
    -- entities using the graph resource have to be declared SUBTYPE of
    -- this entity or any of its subtypes.
    node_1: Node;
      -- one node in the relationship, not necessarily the "first" or
      -- "primary" or "parent".
    node_2: Node;
      -- the other node in the relationship, not necessarily the
      -- "second" or "secondary" or "child".
  END_ENTITY;


  ENTITY Directed_arc
      ABSTRACT SUPERTYPE
      SUBTYPE OF (Arc);
    -- A <Directed_arc> represents a relationship between two <Node>s,
    -- in which case one node is viewed at as being the "first" or
    -- "primary" or "parent" node, the other node is by definition the
    -- "second" or "secondary" or "child" node.
  DERIVE
    start_node: Node := SELF\Arc.node_1;
      -- the "first" or "primary" or "parent" node.
    end_node: Node := SELF\Arc.node_2;
      -- the "second" or "secondary" or "child" node.
  END_ENTITY;


  ENTITY Graph
     ABSTRACT SUPERTYPE
     SUBTYPE OF (Topology);
    -- A <Graph> is defined as a related set of nodes and arcs. The
    -- following rules and constraints apply:
```

```
  -- 1. All nodes of the <Arc>s have to be nodes in <Node>s.
  -- 2. There may be nodes in <Node>s, which are not referred
  --    to by any arc in the set of <Arc>s.
  nodes: SET OF Node;
  arcs: SET OF Arc;
DERIVE
  unconnected_nodes: SET OF Node := find_unconnected_node(nodes,arcs);
  -- the nodes which are not referred to in any arc in the set of
  -- <Arc>s.
WHERE
  no_lost_arc_nodes: graph_condition_on_nodes(nodes, arcs);
    -- all nodes of the <Arc>s have to be nodes in <Node>s.
END_ENTITY;


FUNCTION find_unconnected_node(
  nodes: SET OF Node;
    -- the nodes to be checked.
  arcs: SET OF Arc
    -- the arcs to be checked.
  ) : SET OF Node;
  -- The function <find_unconnected_node> returns the subset of nodes
  -- out of the input set of nodes which are not referred to by any of
  -- the arcs in the input set of arcs.

  RETURN (nodes);
END_FUNCTION;

FUNCTION graph_condition_on_nodes(
  nodes: SET OF Node;
    -- the nodes of a graph to be checked.
  arcs: SET OF Arc
    -- the arcs of a graph to be checked. Both arcs and nodes shall
    -- together describe the same graph.
  ) : BOOLEAN;
  -- The function <graph_condition_on_nodes> returns TRUE if all nodes of
  -- input set of arcs are present in the input set of nodes.

  RETURN(TRUE);
END_FUNCTION;


ENTITY Connected_graph
  ABSTRACT SUPERTYPE
  SUBTYPE OF (Graph);
  -- A <Connected_graph> is a <Graph> where the following
  -- constraints apply:
  -- 1. all arcs have to be connected to form one single structure.
  -- 2. the set of unconnected nodes shall be empty.
  WHERE
    is_connected:   graph_is_connected(SELF\Graph.arcs);
    -- all arcs have to be connected to form one single structure.
    no_lost_nodes: SIZEOF(SELF\Graph.unconnected_nodes) = 0;
    -- the set of unconnected nodes shall be empty.
END_ENTITY;


FUNCTION graph_is_connected(
  arcs: SET OF Arc
    -- the arcs to be checked.
  ) : BOOLEAN;
  -- The function <graph_is_connected> returns TRUE if the arcs in the input
  -- set of arc form a single "structure".

  RETURN(TRUE);
   -- dummy to stop the compiler complain about a missing return statement

END_FUNCTION;


ENTITY Acyclic_graph
  ABSTRACT SUPERTYPE
  SUBTYPE OF (Connected_graph);
  -- An <Acyclic_graph> is a <Connected_graph> with NO cycles.
  WHERE
    no_cycles: TRUE;
  -- graph_has_no_cycles(SELF\arcs);
  -- this graph shall be acyclic.
END_ENTITY;


ENTITY Directed_graph
   ABSTRACT SUPERTYPE
   SUBTYPE OF (Graph);
-- A <Directed_graph> is a <Graph> where all arcs are directed.
```

```
  SELF\Graph.arcs : SET OF Directed_arc;
      -- all arcs of this graph shall be directed.
  END_ENTITY;


  ENTITY Single_entry_directed_graph
     ABSTRACT SUPERTYPE
     SUBTYPE OF (Directed_graph);
  -- A <Single_entry_directed_graph> is a <Directed_graph> which has
  -- exactly one start node referred to by only one arc.
  DERIVE
     entry_node: Node := find_entry_node(SELF)[0];
  WHERE
    entry_node_condition: (SIZEOF(find_entry_node(SELF)) = 1);
      -- this graph shall have exactly one start node referred to by
      -- only one arc.
  END_ENTITY;


  FUNCTION find_entry_node (
    grph: Graph
      -- the graph to find the entry node of.
    ) : SET OF Node;
    -- The function <find_entry_node> returns the number of entry
    -- nodes of graph.
   LOCAL
      node_set: SET OF Node;
   END_LOCAL;

   RETURN (node_set);
     -- dummy to stop the compiler complain about a missing return statement
  END_FUNCTION;


  ENTITY Single_entry_directed_acyclic_graph
     ABSTRACT SUPERTYPE
     SUBTYPE OF (Single_entry_directed_graph, Acyclic_graph);
    -- A <Single_entry_directed_acyclic_graph> is a
    -- <Directed_acyclic_graph> which has exactly one start node
    -- referred to by only one arc.
  END_ENTITY;


  ENTITY Tree
     ABSTRACT SUPERTYPE
     SUBTYPE OF (Single_entry_directed_acyclic_graph);
    -- A <tree> is a <single_entry_directed_acyclic_graph> where no two
    -- arcs end at the same node.
  WHERE
    is_tree: TRUE;
  END_ENTITY;

END_SCHEMA; -- graphs_model.
```

12. Appendix C: Common Application Activity Model

ISO 10303 is the International Standard for the exchange of Product Model Data (STEP). Its objective is to produce a mechanism that is capable of describing product data, throughout the lifecycle of a product, in a form that is unambiguous and independent of any particular software system or hardware platform.

This appendix presents the Shipbuilding Common Application Activity Model (AAM), with the accompanying definitions of the terms describing the information exchanged and the titles of the activities performed. Also included is the node tree that gives an overview of all the pages in the AAM and their hierarchy.

The Common AAM are the higher levels which are common to all the Shipbuilding APs. Each of the APs will contain the common AAM and expand the lower levels where necessary.

The AAM is a graphical representation of the activities carried out in a process and the information flows required between them. The AAM is used to aid identification of these flows of information and therefore clarify the scope and information requirements of the AP. The modelling of this AAM is done using the IDEF-0 (ICAM Definition Language 0) notation.

The application activity model (AAM) in this Annex is intended to provide a common basis from which all of the Shipbuilding Application Protocols can be built and extended according to the individual domain requirements. Such AAMs aid in understanding the scope and information requirements defined for each Application Protocol. This AAM provides the context of all the Application Protocols and as such covers activities which go beyond the scope of them individually. The model is presented as a set of definitions of the activities and the data, and a set of activity figures.

The viewpoint of the application activity model is of an observer of the global ship development process. This activity model identifies the life cycle activities across all shipbuilding APs with extensions and emphasis detailed in each AP where appropriate. Activities relevant to the shipbuilding lifecycle that are do not fit with this activity model but are required and are detailed in other shipbuilding application protocols, should be brought to the attention of the editors of this document so that this generic model can be amended where relevant.

The following definitions describe the activities, inputs, outputs, controls and modifiers which interact as shown in the forthcoming diagrams.

A.1.1 approved design : The approved design is the final design to be submitted as an offer.

A.1.2 arrangements : The arrangements of the ship are the ship’s compartments and spaces. Any description of arrangements will include associated definitions of purpose for the compartment or space.

A.1.3 assemble ship : the activity that assembles the modular units, the serviced parts and additional material that result from the production of steel sub-section. The result is an assembled ship, that still has to be tested.

A.1.4 availability, reliability and maintainability information : The information about the components that is required to install them in the ship and is required for planned maintenance.

A.1.5 basic hull parameters : Estimated principal dimensions based on historical data or preliminary design development.

A.1.6 budget : the cost constraint on the design building and maintenance of the ship.

A.1.7 calculate cost of ship : This activity describes creation of negotiating documents based on technical product data and their estimated manufacturing cost. The results of this activity may contain sale price documents, financing support plan and documents describing funding and possible loans.

A.1.8 certificates : The certificates issued by the Classification Society on completing the ship.

A.1.9 check design against rules and regulations : This is the top level activity for the approval of the primary design as part of the approval and certification process. The content of this activity is the same for all ships when it comes to conformance with Main Class Rules, but varies when it comes to additional class rules (type of vessel) and register notations. The activities performed are tailored to the rule requirements for general arrangement and global strength. This part of the approval is necessary before the yard can start ordering steel.

A.1.10 Classification Society : An organisation that enhances the safety of life and property at sea by providing rules, regulations and personnel for assessing and classifying ships during their lifecycle.

A.1.11 complete and approve design of machinery : The selection, arrangement and approval of the power plant in terms of the main engine, associated propulsion system and its auxiliary machinery.

A.1.12 complete and approve design of outfitting and distribution systems : The selection and approval of the necessary outfitting equipment. The selection is based mainly on former designs and in accordance with the requirements. It also contains the layout of the different types of distribution systems such as piping and HVAC.

A.1.13 complete and approve design of ship structure : The completion and approval of the ship structural design.

A.1.14 complete and approve ship design : The production and approval of ship design product data, documents and the classification drawings using the preliminary design from the bid preparation, as well as the required rules and regulations. The result of this activity is the approved design and the production and delivery schedule.

A.1.15 consultants : Organisations that provide specific services to shipyards, ship owners and classification societies during the ship lifecycle.

A.1.16 contract : The contract is the output from the activity which involves placing the order for the ship. The contract is used as a constraint in subsequent activities such as final design and approval and production.

A.1.17 cost : The calculated cost of the ship based on the cost of material and labour.

A.1.18 create preliminary design : All design activities relevant in a very preliminary stage of ship design in consideration of classification rules, national/international demands, shipyard constraints and owner requirements. The aim of this task is to make a shipyard offer.

A.1.19 create preliminary general arrangements : The activity that produces the preliminary compartmentation plans from the preliminary hull form definition.

A.1.20 create preliminary hull form : The activity that is the first step of designing a ship. Using parent ships main dimensions and form parameters one or more preliminary hull forms will be generated.

A.1.21 create preliminary machinery design : The activity that produces the preliminary designs for the ship machinery; including the prime mover, shaft system, fuel system, power systems and cargo handling equipment.

A.1.22 create preliminary outfitting design : The activity that produces the preliminary design for the ship's outfitting, including distributed systems, such as piping and electrical systems.

A.1.23 create preliminary structure design : The activity that produces the preliminary steel structure design, including the arrangement of the primary structural members.

A.1.24 decide post-sales & maintenance support : The activity that puts together the maintenance package for the ship. This is part of the tender document and includes the post sales support.

A.1.25 decommission and disassemble : All activities relating to the last stage of the ship's lifecycle. It consists of the decommissioning and dismantling of the ship.

A.1.26 design schedule : Data that controls the time from the design phase to production.

A.1.27 distribution and outfitting design : The design of the distribution systems (electrical and piping) and the outfitting.

A.1.28 estimate hydrodynamics and powering : The activity that approximates hydrodynamic properties data calculations such as resistance, propulsion, seakeeping and manoeuvrability for the preliminary hull form.

A.1.29 evaluate request & schedule bid : This describes the activities of the shipyard when evaluating the inquiry of the ship owner for a new ship.

A.1.30 feedback : The outputs from activities which then feed back and modify previous activities in the lifecycle on the current or subsequent ships.

A.1.31 finalise and approve general arrangements : The activity that details the general arrangement after having created a draft layout. The ship's systems are described by a compartment and access drawing showing the location, the access, and the size of the different compartments.

A.1.32 finalise and approve hull form : The activity in which the hull form is finalised from the preliminary design. The result is a final and approved hull form design.

A.1.33 finalise and approve hydrodynamics and powering : This includes all relevant hydrodynamic calculations such as resistance, propulsion, seakeeping and manoeuvrability.

A.1.34 general arrangements : The space arrangement plan from the preliminary design stage.

A.1.35 historical data from previous designs : Data held by the shipyard or model basin on previous ship designs and used to estimate the hydrodynamics, powering requirements and seakeeping.

A.1.36 hull form sections : The design of the hull moulded form at planar sections taken along the longitudinal axis of the ship.

A.1.37 hull moulded form : The definition of the shape of the hull of the ship, resulting from the addition of the aft-body, mid-body and fore-body definitions, which does not take into account the thickness of the material from which the hull is made.

A.1.38 hydrodynamics & powering results : The results of calculations and model basin tests. They contain resistance, propulsion, propeller performance, brake power, service speed, sea keeping and manoeuvrability data.

A.1.39 knowledge and experience : The previous experience and knowledge of companies involved throughout the ship lifecycle.

A.1.40 laws, rules and regulations : National laws, statutory regulations and classification society rules that are used to control the design, manufacture, operation, maintenance and scrapping of the ship.

A.1.41 list of required certificates : The result of placing an order, this is the list supplied by the owner for certificate requirements.

A.1.42 loading and stability manual : a booklet which is placed on board the ship for the information of the master, which enables him or her to load the ship within prescribed limits, relating to strength and stability.

A.1.43 machinery design : The design drawings and electronic models of the ship mechanical systems. An output from the final design process.

A.1.44 machinery weights : These outputs are the results of several calculation and design activities which result in an estimated weight for all machinery.

A.1.45 manufacturing restrictions : A constraint on the ship construction and design processes governed by available technology and shipyard facilities.

A.1.46 material list : The list of raw materials needed to manufacture the ship. A result of the final design process.

A.1.47 modifications from machinery : Modifications to the hydrodynamics and powering due to feedback from the preliminary machinery design.

A.1.48 modifications to hull form : Modifications to the hull shape due to feedback from hydrodynamics and powering results and the final design process.

A.1.49 modular units : sub-sections of the ship complete with machinery and outfitting which will be assembled to create the final product.

A.1.50 offer : The result of the preliminary design process. It will contain the shipyard's data for producing the requested ship.

A.1.51 offer guidelines : The offer guidelines include the data necessary to make an unconditional offer to the ship owner

A.1.52 operate and maintain a ship : The activity that describes the running and maintenance of the ship during its service lifetime.

A.1.53 operational information : Accumulated information during the operation phase of the ship used for maintenance and in the final scrapping stage.

A.1.54 owner : The organisation which requests, orders, takes delivery of and, for the purposes of this model, operates the ship.

A.1.55 owner request, requirements : The requirements document that is submitted to the shipyard by the owner upon the invitation to tender.

A.1.56 perform ship lifecycle : All of the lifecycle activities associated with a ship.

A.1.57 place order : The owner places an order for a ship from the bids that have been submitted. From this a contract is awarded.

A.1.58 planned maintenance system : Data created during the final design process and used during the operation and maintenance of the ship.

A.1.59 pre layout : The very initial layout of the ship which is produced during the bid evaluation stage and is the basis for the preliminary design.

A.1.60 preliminary design : The preliminary design is that which is completed in the phases leading up to the submission of the tender.

A.1.61 preliminary general arrangements : The definition of the ship general arrangements as a result of the preliminary design process.

A.1.62 preliminary hull form : The definition of the hull form, as a result of the preliminary design process. Used in the offer documents and for preliminary compartment design, hydrodynamics and powering calculations.

A.1.63 preliminary machinery design : The definition of the ship mechanical systems. Used early to estimate the noise, speed and vibration and to estimate the machinery weights.

A.1.64 preliminary machinery, structure and outfitting design : Feedback consisting of the preliminary designs for machinery, structure and outfitting and furnishing. This allows the creation of preliminary general arrangements.

A.1.65 preliminary outfitting design : The definition of the ship's outfitting and accommodation, resulting from the preliminary design process.

A.1.66 preliminary structure design : The definition of the preliminary ship structure during the preliminary design process.

A.1.67 prepare bid : This activity includes all activities of the yard regarding preparation and submission of the offer to the ship owner for the ship to be built.

A.1.68 present offer : The activity concerned with presentation of the offer to build the ship to the prospective ship owner.

A.1.69 produce and approve reference documents : the technical documentation for the ship is produced using production information. The output includes the loading and stability manual which is approved by the Classification Society.

A.1.70 produce and inspect a ship : This activity includes high-level activities such as produce, monitor and inspect ship production. Inspect, means the controlling of all activities throughout the whole production life cycle of a ship.

A.1.71 produce modular build units : this activity covers the production of the modular units which will make up the completed ship. They are produced from the steel-subsections and their production is controlled by the schedule, contract, the approved design, and any manufacturing restrictions. The results of the activity are the modular units which are assembled into the ship.

A.1.72 produce steel sub-sections : this activity covers the production of the steel subsections which make up the structure of the completed ship. This is controlled by the schedule, contract, the approved design, and any manufacturing restrictions.

A.1.73 product component information : The technical data about the components that will be incorporated into the ship. These are taken into consideration when the preliminary designs are being made.

A.1.74 production and delivery schedule : The schedule according to which the ship is maufactured and delivered.

A.1.75 production information : information describing a product, e.g. dimensions, mechanical properties, workshop information.

A.1.76 propeller design : The design of the propeller or propulsor as a result of the hydrodynamics and powering calculations. The design controls some of the machinery design activity.

A.1.77 quality assurance : the rules applied by an organisation within the shipyard that has the task to audit the shipyard organisation and applied processes in a manner such that the quality of the resulting product is assured.

A.1.78 request a ship : The first activities of a ship owner when intending to order a ship. Having definite ideas regarding appearance and functionality of the ship, the owner expresses these ideas in an inquiry to the shipyard.

A.1.79 request for production changes : Changes that are requested to the ship design as a result of production experience or difficulties with the realisation of the ship design.

A.1.80 resistance and shaft power : The result of the activity to estimate hydrodynamics and powering. Resistance and shaft power is a constraint on the creation of the preliminary hull form.

A.1.81 resources : The shipyard, classification society, and outside consultants.

A.1.82 schedule : The schedule is formed as a part of the final design process. It governs the timing of the production phases.

A.1.83 scrapping plan : The document used to schedule the time and resources required to dismantle the ship.

A.1.84 ship : a large waterborne vessel whose design, manufacture and lifecycle operation is governed by the principles of naval architecture and in accordance with international and classification society regulations.

A.1.85 ship product model data : The product data of the accumulated throughout its lifecycle. Because scrapping is part of the lifecycle the ship is not an output, only the documented information and knowledge about the ship survives.

A.1.86 ship weight modifications : Modifications to ship weight due to the preliminary structure design. This is fed back to modify the preliminary hull form and revise the preliminary general arrangements.

A.1.87 shipyard : An organisation that designs, builds, maintains, and repairs ships.

A.1.88 specify ship : All activities associated with the production of a detailed specification of the ship prior to a contract being placed.

A.1.89 steel sub-sections : the sub-sections of the steel structure which are outfitted with the machinery and distribution systems before assembly.

A.1.90 structural design : The design of the hull structure including hull, bulkheads, decks and stiffeners.

A.1.91 technical documentation : In case of maintenance the technical documentation of a system means part of the product description required to perform preventative maintenance, repair and failure analysis of that system. Technical information is an output which includes more detail information about material parts needed for producing the ship/system.

A.1.92 technical requirements : The owner's specifications that must be realised by the completed ship.

A.1.93 test results : maintenance test results are the results of functional tests carried out after the execution of maintenance actions.

A.1.94 test ship : this activity tests the actual ship against the design, contract and rules and regulations. The structure, is tested and sea trials are carried out. The test results are an output from this activity.

A.1.95 test structures : the steel structures are tested against rules and regulations and the design. The output is the test result documentation.

A.1.96 test systems : the ship's systems including outfitting, machinery and mission systems are tested against rules and regulations and the design. The output is the test result documentation.

A.1.97 transportation need : A constraint which determines the specification for the ship construction.

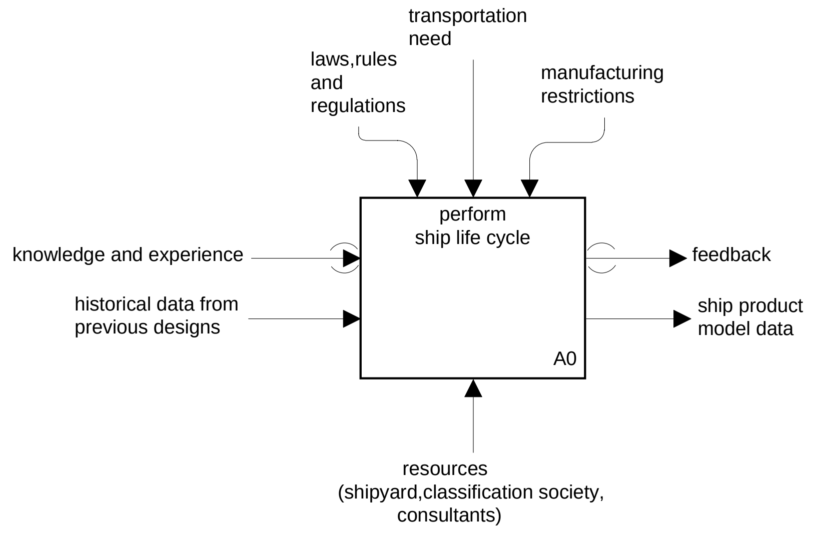
A.1.98 weights and centres of gravity : Weights and centres of gravity necessary for further calculations.

A.1.99 workload : The total effort required to build the chosen ship design as estimated by the shipyard and assisting consultants.

Activity Node Tree

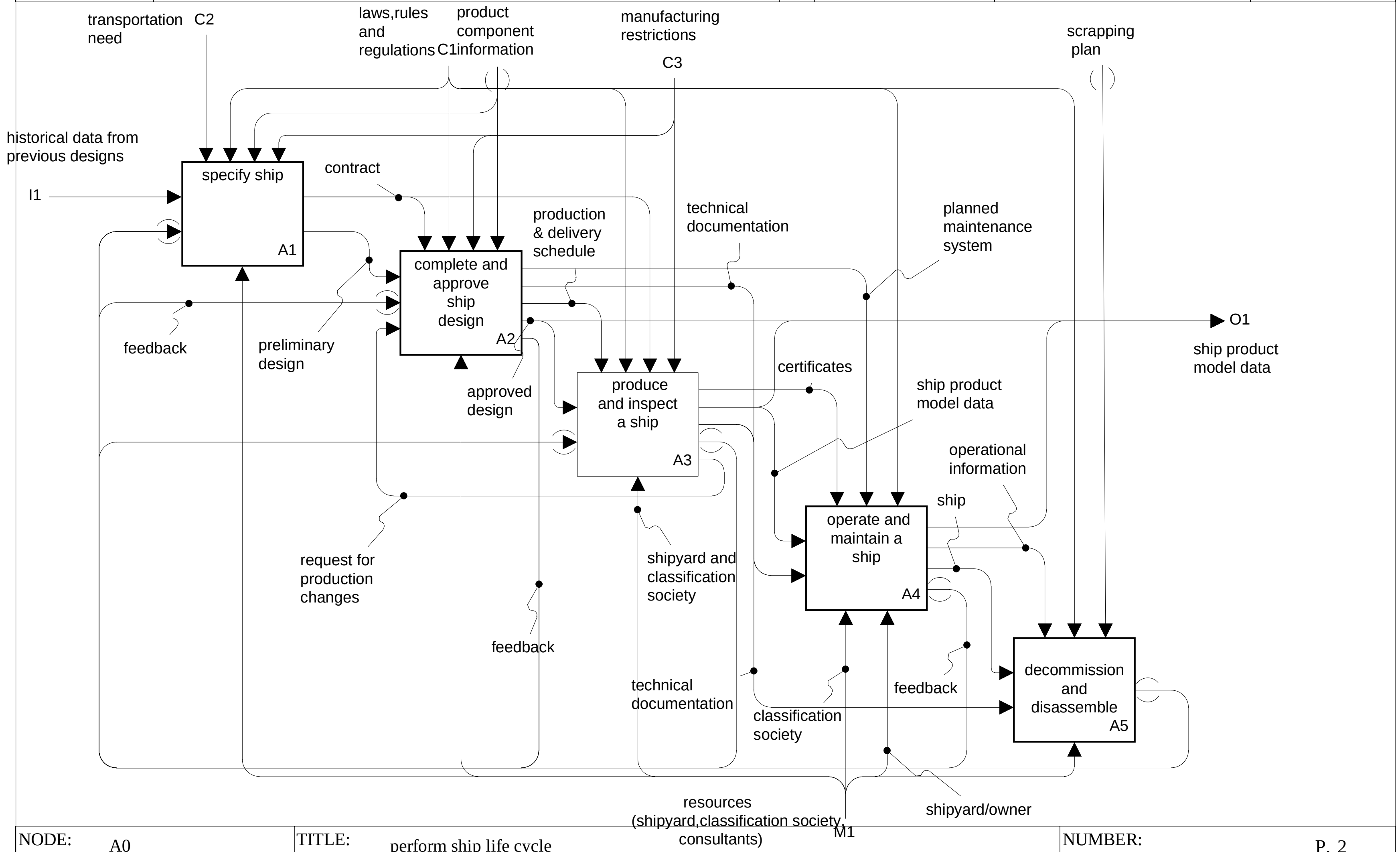
- [A0] perform ship life cycle
  - [A1] specify ship
    - [A11] request a ship
    - [A12] prepare bid
      - [A121] evaluate request & schedule bid
      - [A122] create preliminary design
        - [A1221] create preliminary hull form
        - [A1222] create preliminary general arrangements
        - [A1223] estimate hydrodynamics and powering
        - [A1224] create preliminary structure design
        - [A1225] create preliminary machinery design
        - [A1226] create preliminary outfitting design
      - [A123] decide post-sales & maintenance support
      - [A124] calculate cost of ship
      - [A125] present offer
    - [A13] place order
  - [A2] complete and approve ship design
    - [A21] finalise and approve general arrangements
    - [A22] finalise and approve hull form
    - [A23] finalise and approve hydrodynamics and powering
    - [A24] complete and approve design of ship structure
    - [A25] complete and approve design of machinery
    - [A26] complete and approve design of outfitting and distribution sy
  - [A3] produce and inspect a ship
    - [A31] produce steel sub-sections
    - [A32] produce modular build units
    - [A33] assemble ship
    - [A34] test ship
      - [A341] test structures
      - [A342] test systems
      - [A343] conduct contractor sea trials
      - [A344] conduct acceptance trials
    - [A35] produce and approve reference documents
  - [A4] operate and maintain a ship
  - [A5] decommission and disassemble

USED AT:	EDITOR: John Kendall DATE: 22/05/96 PROJECT: STEP Shipbuilding Common AAM REV: 2.0 NOTES: 1 2 3 4 5 6 7 8 9 10	x	WORKING	READER DATE	CONTEXT: Top
			DRAFT		
			RECOMMENDED		
			PUBLICATION		


transportation need
laws,rules and regulations
manufacturing restrictions
perform ship life cycle
A0
knowledge and experience
historical data from previous designs
feedback
ship product model data
resources (shipyard,classification society, consultants)

NODE: A-0	TITLE:	NUMBER: P. 1

USED AT:	EDITOR: John Kendall PROJECT: STEP Shipbuilding Common AAM NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: 22/05/96 REV: 2.0	x	WORKING	READER DATE	CONTEXT:
				DRAFT		
				RECOMMENDED		
				PUBLICATION		

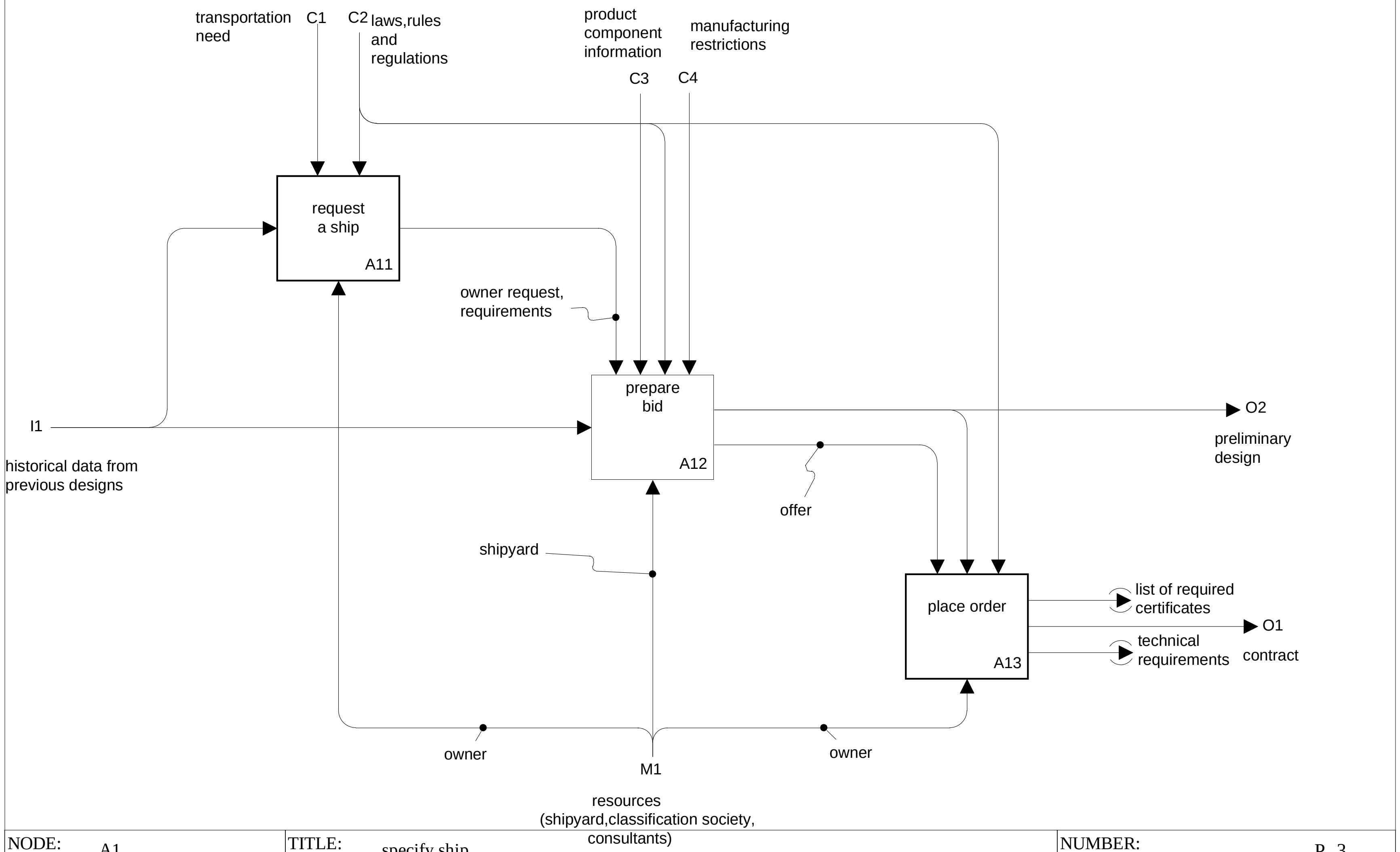


transportation need C2
laws,rules and regulations C1
product component information
manufacturing restrictions
C3
scrapping plan
historical data from previous designs
I1
specify ship
A1
contract
complete and approve ship design
A2
production & delivery schedule
technical documentation
planned maintenance system
O1
ship product model data
feedback
preliminary design
approved design
produce and inspect a ship
A3
certificates
ship product model data
operational information
ship
operate and maintain a ship
A4
request for production changes
shipyard and classification society
feedback
technical documentation
classification society
feedback
decommission and disassemble
A5
resources (shipyard,classification society, consultants)
M1
shipyard/owner

NODE: A0	TITLE: perform ship life cycle	NUMBER: P. 2

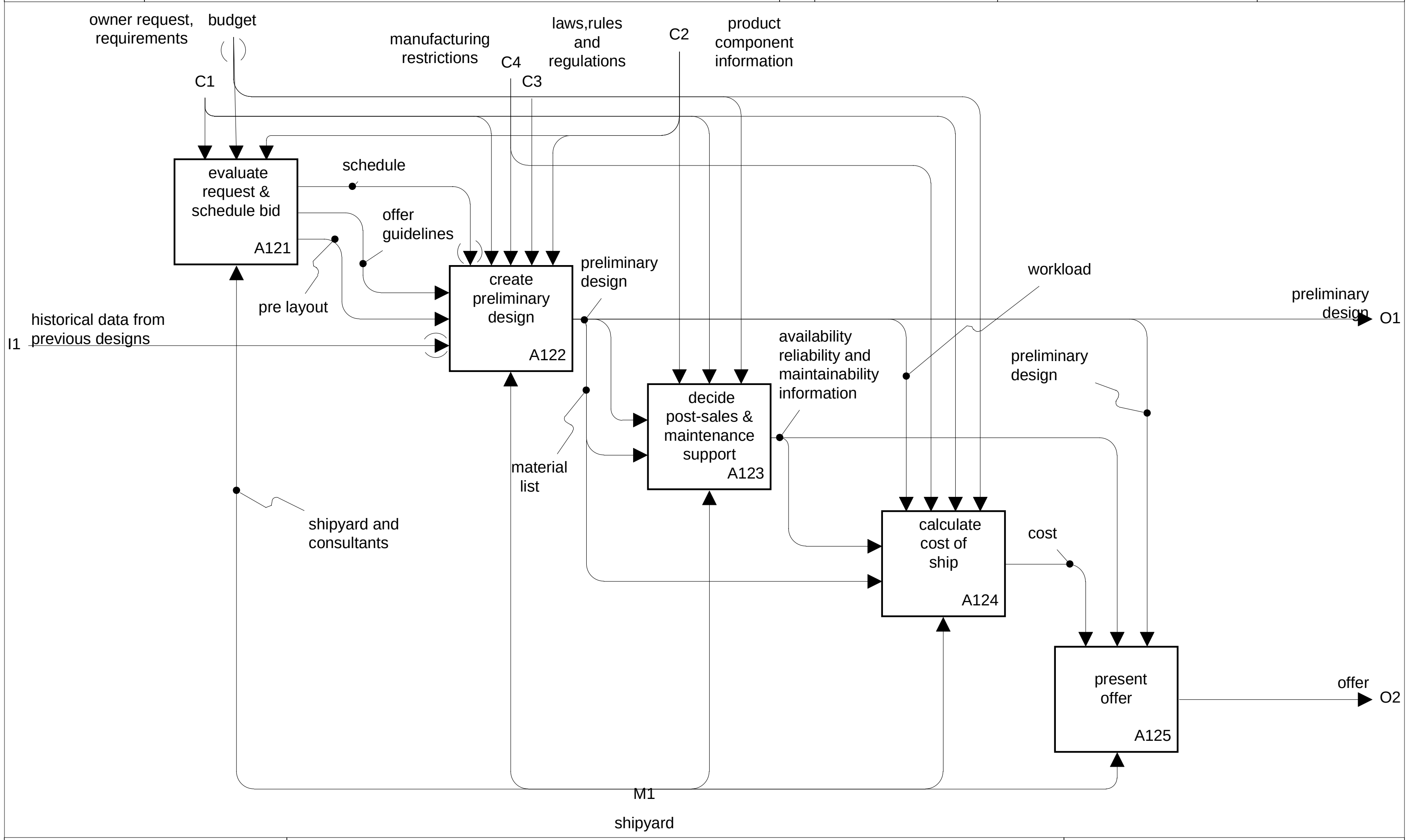
USED AT:	EDITOR: John Kendall PROJECT: STEP Shipbuilding Common AAM NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: 22/05/96 REV: 2.0	x	WORKING	READER	DATE
				DRAFT		
				RECOMMENDED		
				PUBLICATION		

CONTEXT: ■ □ □ □ □


transportation need
C1
C2
laws,rules and regulations
product component information
manufacturing restrictions
C3
C4
request a ship
A11
owner request, requirements
prepare bid
A12
I1
historical data from previous designs
O2
preliminary design
offer
shipyard
place order
A13
list of required certificates
O1
technical requirements
contract
owner
owner
M1
resources (shipyard,classification society, consultants)

NODE: A1	TITLE: specify ship	NUMBER: P. 3

USED AT:	EDITOR: John Kendall	DATE: 22/05/96	x	WORKING	READER DATE	CONTEXT:
	PROJECT: STEP Shipbuilding Common AAM	REV: 2.0		DRAFT		
				RECOMMENDED		
	NOTES: 1 2 3 4 5 6 7 8 9 10			PUBLICATION		

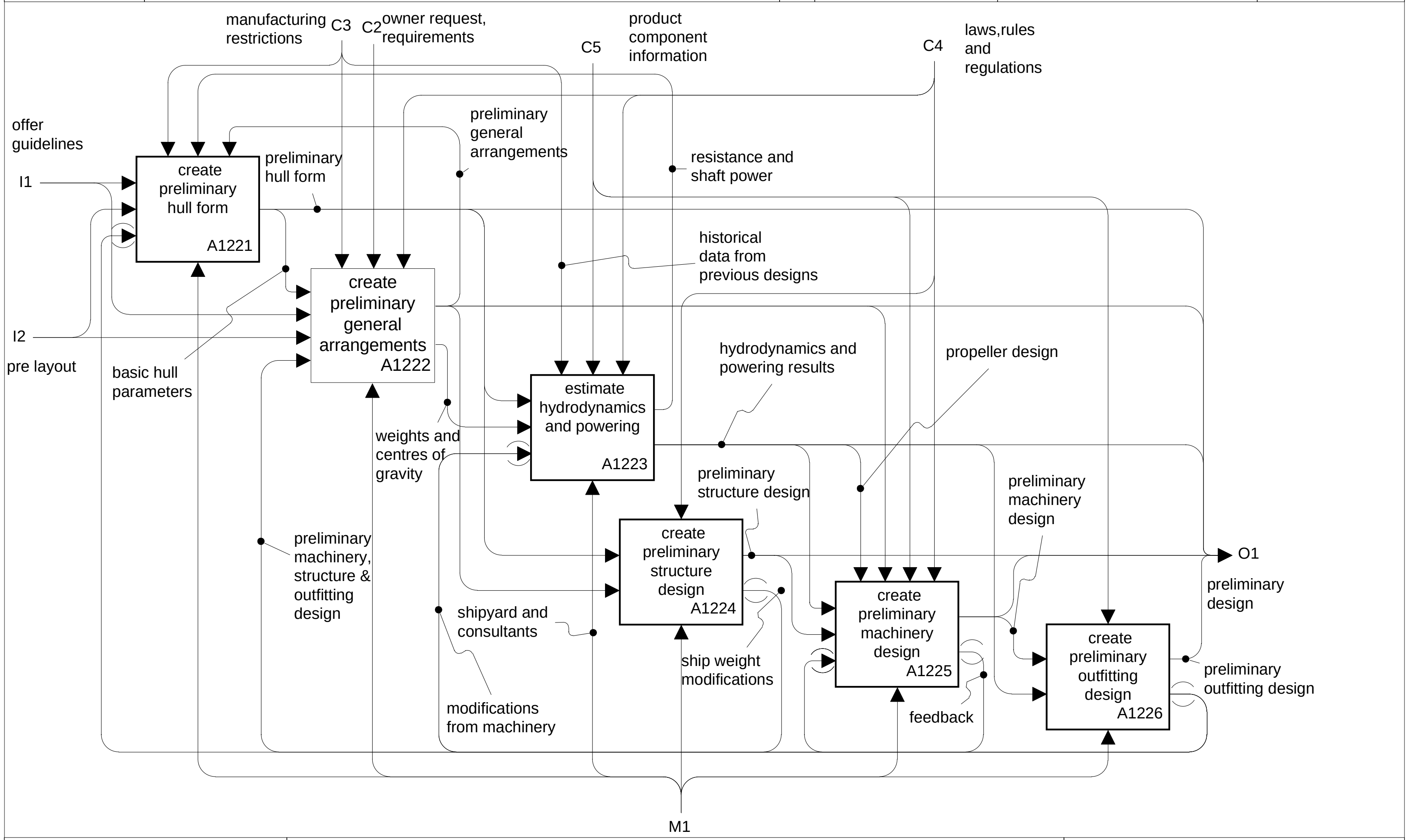


owner request, requirements
budget
C1
manufacturing restrictions
C4
C3
laws,rules and regulations
C2
product component information
evaluate request & schedule bid
A121
schedule
offer guidelines
pre layout
create preliminary design
A122
preliminary design
historical data from previous designs
I1
workload
preliminary design
O1
decide post-sales & maintenance support
A123
availability reliability and maintainability information
preliminary design
material list
shipyard and consultants
calculate cost of ship
A124
cost
present offer
A125
offer
O2
M1
shipyard

NODE: A12	TITLE: prepare bid	NUMBER: P. 4

USED AT:	EDITOR: John Kendall PROJECT: STEP Shipbuilding Common AAM NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: 22/05/96 REV: 2.0	x	WORKING	READER DATE
				DRAFT	
				RECOMMENDED	
				PUBLICATION	

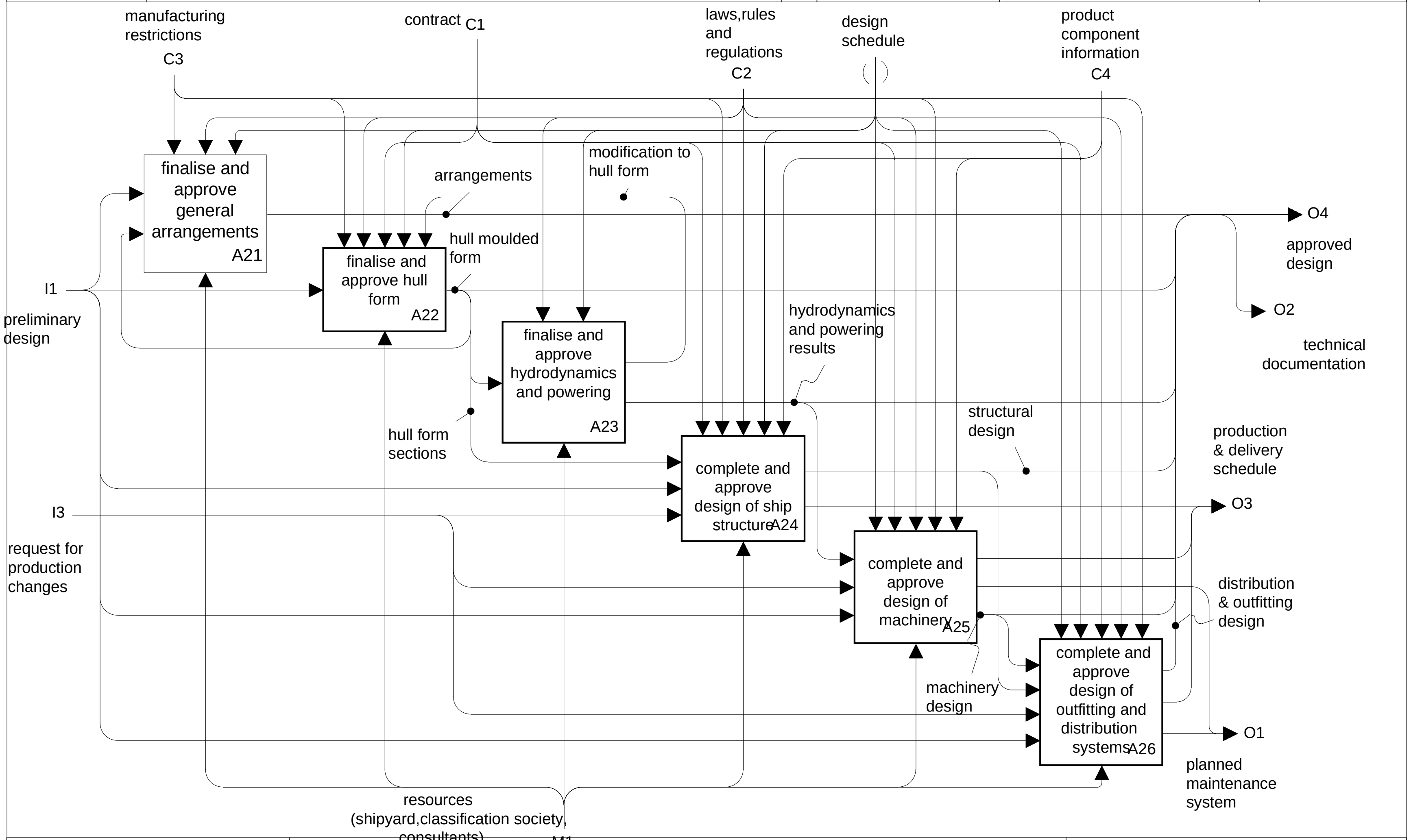
CONTEXT:



manufacturing restrictions
C3
C2
owner request, requirements
C5
product component information
C4
laws,rules and regulations
offer guidelines
I1
create preliminary hull form
A1221
preliminary hull form
preliminary general arrangements
resistance and shaft power
historical data from previous designs
create preliminary general arrangements
A1222
I2
pre layout
basic hull parameters
estimate hydrodynamics and powering
A1223
hydrodynamics and powering results
propeller design
weights and centres of gravity
preliminary structure design
preliminary machinery design
preliminary machinery, structure & outfitting design
create preliminary structure design
A1224
O1
preliminary design
create preliminary machinery design
A1225
create preliminary outfitting design
A1226
preliminary outfitting design
shipyard and consultants
ship weight modifications
feedback
modifications from machinery
M1

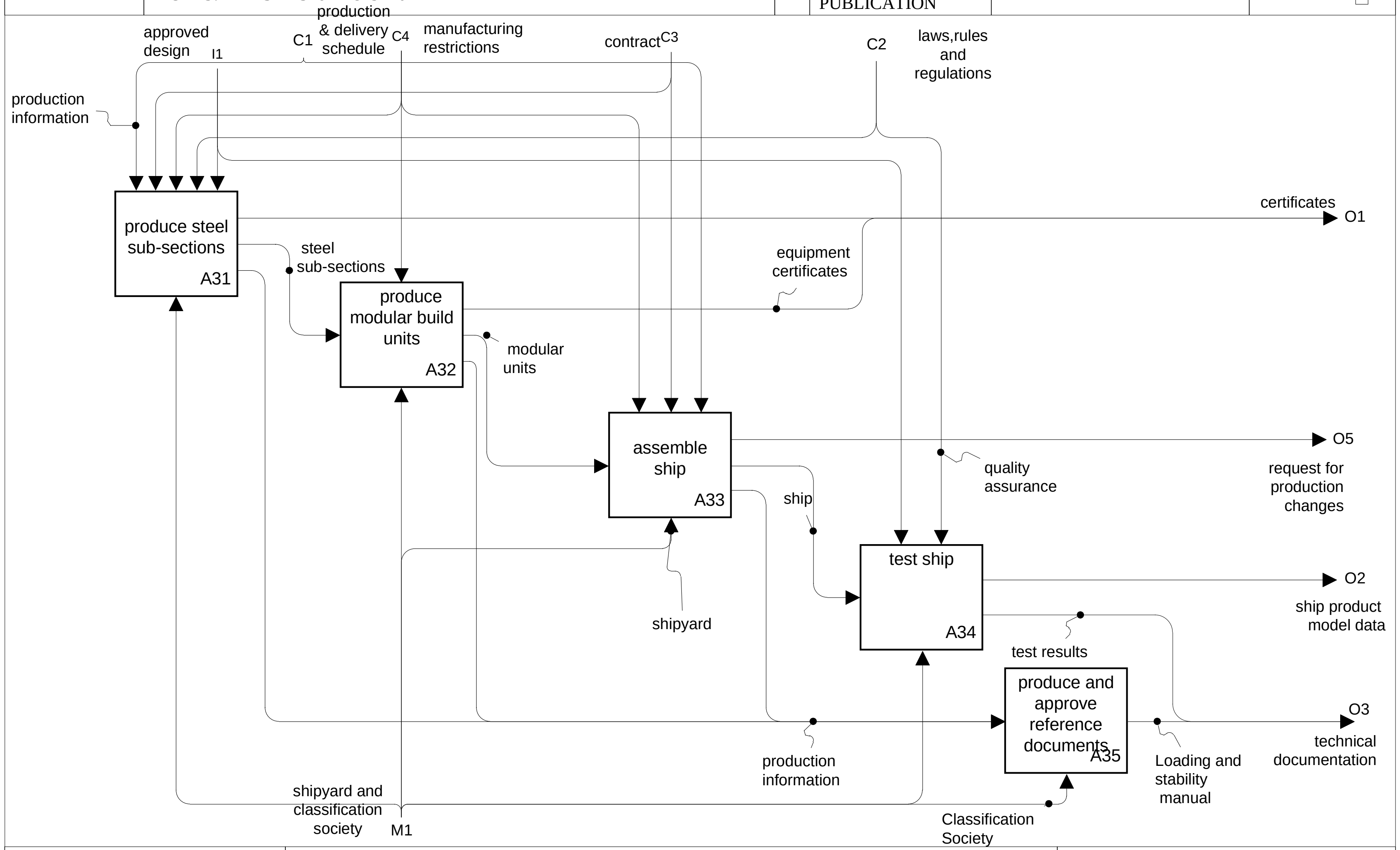
NODE: A122	TITLE: create preliminary design shipyard	NUMBER: P. 5

USED AT:	EDITOR: John Kendall PROJECT: STEP Shipbuilding Common AAM NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: 22/05/96 REV: 2.0	x WORKING DRAFT RECOMMENDED PUBLICATION	READER DATE	CONTEXT:


manufacturing restrictions
C3
contract C1
laws,rules and regulations
C2
design schedule
product component information
C4
finalise and approve general arrangements
A21
arrangements
modification to hull form
finalise and approve hull form
A22
hull moulded form
O4
approved design
I1
preliminary design
finalise and approve hydrodynamics and powering
A23
hydrodynamics and powering results
O2
technical documentation
hull form sections
structural design
production & delivery schedule
complete and approve design of ship structure
A24
I3
request for production changes
O3
complete and approve design of machinery
A25
distribution & outfitting design
machinery design
complete and approve design of outfitting and distribution systems
A26
O1
planned maintenance system
resources (shipyard,classification society, consultants)
M1

NODE: A2	TITLE: complete and approve ship design	NUMBER: P. 6

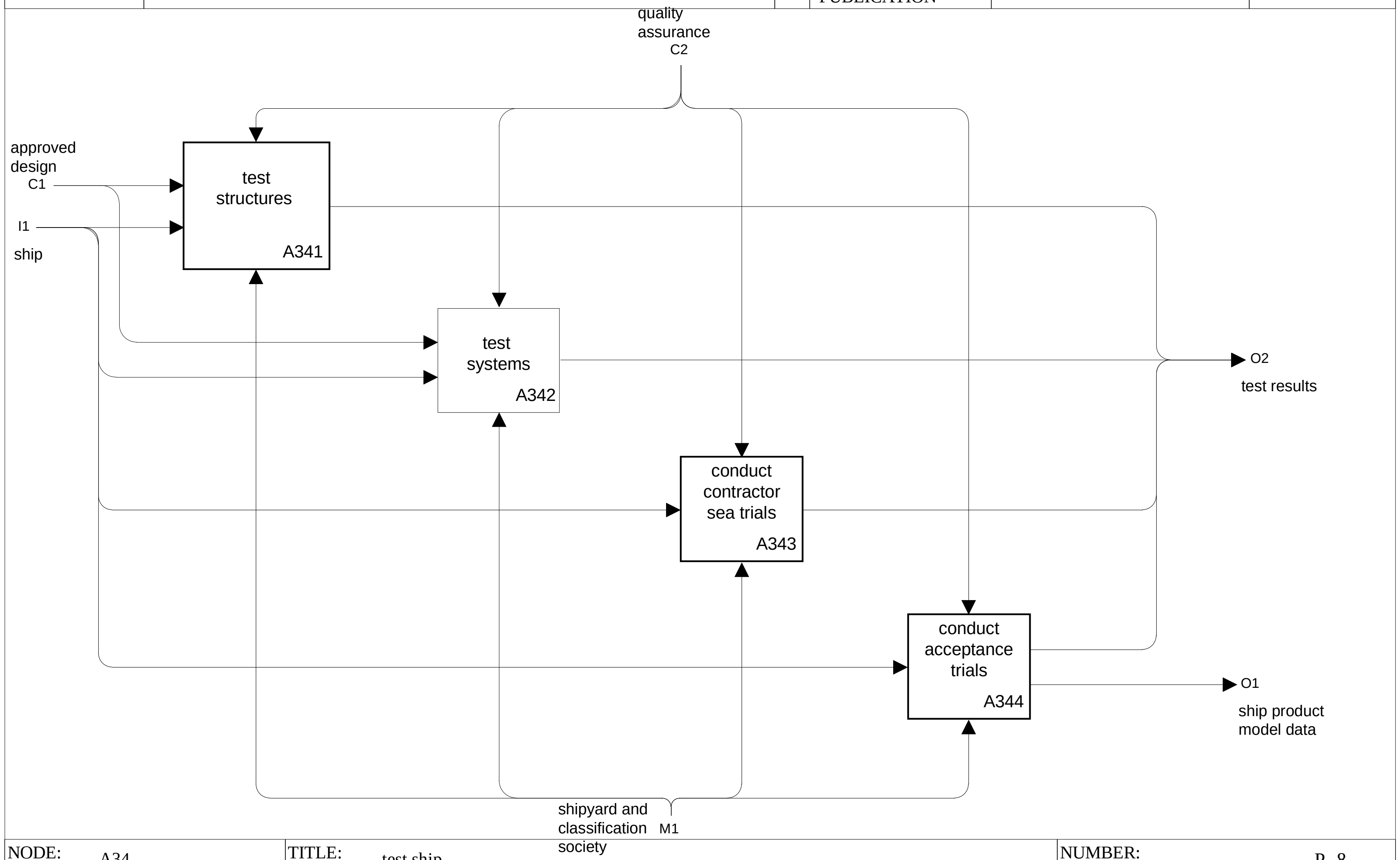
USED AT:	EDITOR: John Kendall PROJECT: STEP Shipbuilding Common AAM NOTES: 1 2 3 4 5 6 7 8 9 10	DATE: 22/05/96 REV: 2.0	x	WORKING	READER DATE	CONTEXT:
				DRAFT		
				RECOMMENDED		
				PUBLICATION		



NODE: A3	TITLE: produce and inspect a ship	NUMBER: P. 7

USED AT:	EDITOR: John Kendall	DATE: 22/05/96	x	WORKING	READER DATE
	PROJECT: STEP Shipbuilding Common AAM	REV: 2.0		DRAFT	
				RECOMMENDED	
	NOTES: 1 2 3 4 5 6 7 8 9 10			PUBLICATION	

CONTEXT:



NODE: A34	TITLE: test ship	NUMBER: P. 8